P.1877/86

5 1986

informatyka

Prof. Dines Björner o definicji formalnej Ady

Mikrokomputerowy system operacyjny SI

Programowanie logiczne

WYDAWNICTWO NOT SIGMA

Cena 120 zł

informatyka

5 1986

Nr 5

Miesięcznik Rok XXI

Maj 1986

Organ Komitetu Informatyki MNSZWiT oraz Komitetu Naukowo-Technicznego NOT ds. Informatyki

KOLEGIUM REDAKCYJNE:

Dr inż. Wacław ISZKOWSKI, mgr Teresa JABŁOŃSKA (sekretarz redakcji), Władysław KLEPACZ (redaktor naczelny), dr inż. Marek MACHURA, Maria PAWLAK (sekretarz redakcji), dr inż. Wiktor RZECZKOWSKI, mgr inż. Jan RYŻKO, dr inż. Janusz ZALEWSKI (zastępca redaktora naczelnego)

STALE WSPÓŁPRACUJĄ:

Mgr inż. Witold ABRAMOWICZ (Szwajcaria), mgr inż. Teresa WILCZEK

PRZEWODNICZĄCY RADY PROGRAMOWEJ:

Prof. dr hab. Juliusz Lech Kulikowski

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 00-041 Warszawa, ul. Jasna 14/16, pok. 243 i 244, tel. 27-71-40 lub 26-82-61 w. 184

Zakł. Graf. „Tamka". Zam. 0346-1300/86. Obj. 4,0 ark. druk. Nakład 8300 egz. P-70.

ISSN 0542-9951, INDEKS 36124

Cena egzemplarza 120 zł
Prenumerata roczna 1440 zł

WYDAWNICTWO NACZELNA ORGANIZACJA TECHNICZNA

SIGMA

CZASOPISM I KSIĄŻEK TECHNICZNYCH

00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE

	Strona
Rola i zakres formalnej definicji Ady (1) *Dines Bjørner*	1
Mikrokomputerowy system operacyjny S1 (1) *Krzysztof Perycz*	5
Programowanie logiczne *Krzysztof Michalik*	7
Norma IEEE — języki asemblerowe dla mikroprocesorów *Piotr Cofta*	11
Język programowania ICON (3) *Jerzy Karczmarczuk*	14
Mechanizm półautomatycznego generowania poleceń dla systemu operacyjnego RSX-11M *Paweł Grzegorzewicz*	17
Środki do przetwarzania tekstów — historia i dzień dzisiejszy *Andrzej Maciol, Adam Stawowy*	20
ZE ŚWIATA	25
Mikrokomputery w Wielkiej Brytanii	
TERMINOLOGIA	
Zapożyczenia w polskim słownictwie informatycznym	27
POGLĄDY	28
Zakleszczenie	

W NASTĘPNYCH NUMERACH

- Władysław Udrycki o MML — języku komunikacji z systemem komutacyjnym
- Zbyszko Królikowski o generowaniu planów wykonywania transakcji w systemach rozproszonych baz danych
- Zdzisław Płoski o LOGO Sinclaira ze stacją dyskietek
- Zbigniew Szkaradnik o grafice żółwia w języku Forth
- Prof. Vasilii Zakharow o komputerach osobistych i indywidualnych stanowiskach roboczych
- Cykl artykułów o IBM PC

DINES BJØRNER
Dansk Datamatik Center
Lingby
Dania

P. 1877/86

Rola i zakres formalnej definicji Ady (I)

Cel podjęcia prac nad opracowaniem formalnej definicji Ady jest dwojaki [3].

Po pierwsze — chodzi o zidentyfikowanie:
- kwestii dotyczących definicji związanych z językiem programowania,
- użytkowników specyfikacji języka,
- technik specyfikacji.

Po drugie — chodzi o określenie wpływu tych czynników na wymagania stawiane formalnej definicji Ady.

Poniżej przedstawiono trzy wymienione — prawie od siebie niezależne — aspekty definicji formalnej oraz omówiono ich znaczenie dla sformułowania odpowiednich wymagań.

KWESTIE JĘZYKOWE

Istnieje szereg kwestii zwanych językowymi, które należy wziąć pod uwagę przy opracowywaniu definicji formalnej. Mogą one dotyczyć projektu języka, jego właściwości, użycia i implementacji, normalizacji, nauczania i prowadzenia prac badawczych.

Projektant języka bierze zwykle pod uwagę dwa czynniki: techniki programowania (metodologię) i implementację translatora. Powinien on jednak uwzględniać także łatwość i elegancję opisu semantyki. Pod tym względem specyfikacja formalna może stanowić użyteczne narzędzie dla projektanta. Inną istotną kwestią jest udzielenie odpowiedzi na pytanie: w jakim zakresie i w jakim sensie projekt języka umożliwia tworzenie jego podzbiorów i rozszerzeń.

Podstawowa kwestia dotycząca właściwości języka polega na takim scharakteryzowaniu i sklasyfikowaniu tych właściwości, aby projekt i jego specyfikacja możliwie najbardziej bezpośrednio i najwierniej definiowały te cechy, niezależnie od szczegółowego opisu semantyki. Do właściwości konstrukcji językowych, które bierze się pod uwagę, należą:

- **cechy deterministyczne,** jak kolejność wykonywania instrukcji i porządek opracowywania wyrażeń (np. od strony lewej do prawej),
- **cechy niedeterministyczne,** jak dowolność w kolejności obliczania np. parametrów podprogramów,
- **współbieżność** (równoległość), jak np. wielozadaniowość,
- **niepoprawność,** polegająca na tym, że niektóre konstrukcje poprawnie złożone syntaktycznie nie są zdefiniowane semantycznie,
- **niezdefiniowania** (ang. undefinedness),
- **cechy zależne od implementacji.**

Problem polega na sklasyfikowaniu wszystkich konstrukcji lub ich kombinacji, zgodnie z powyższym schematem, oraz na znalezieniu i zastosowaniu najbardziej odpowiedniej techniki definiowania.

Implementacja języka polega na tworzeniu jego interpreterów, kompilatorów i narzędzi wspomagających. Interpretacja dotyczy zakresu, w jakim poszczególne wiązania w programie (wiążące nazwy z ich znaczeniem) mogą być wykonane tylko w czasie wykonania. Znaczy to, że im bardziej dynamiczne są takie wiązania, w sensie wiązania nazw z różnymi rodzajami (typami) obiektów, tym więcej programów można poddać interpretacji. Na przeciwległym biegunie znajdują się wiązania całkowicie statyczne, tj. takie, których ważność można sprawdzić przed czasem wykonania, w tzw. czasie kompilacji. Im więcej jest tych wiązań, tym więcej programów można skompilować. Tak więc istotne jest pytanie: czy definicja języka odzwierciedli to zróżnicowanie? Ważne jest również, w jakim stopniu sam język nadaje się do tworzenia narzędzi wspomagających programowanie.

Języka używa się do programowania i dokumentowania programów w różnych dziedzinach zastosowań, jak: obliczania numeryczne i symboliczne, systemy sterowania i systemy łączności, bazy danych i sztuczna inteligencja. W każdej z tych dziedzin użytkownicy oczekują, że definicja języka dostarczy im odpowiedzi na pytania związane z jego stosowaniem. Ogólne problemy dotyczą tego, w jakim stopniu język programowania nadaje się do stosowania określonego podejścia (np. kolejnych uściśleń lub modularyzacji programu) w wytwarzaniu oprogramowania i do dowodzenia poprawności programów, oraz jakie mechanizmy języka decydują o jego przydatności do dokumentowania programów.

Do istotnych kwestii związanych z normalizacją języka należą: łatwość utworzenia normy i stosowania się do niej oraz łatwość jej pielęgnacji. W nauczaniu języka istotną rolę odgrywa łatwość jego rozumienia i pisania podręczników.

Wyliczenie tych kwestii językowych nie oznacza jeszcze, że liczą się one wszystkie w opracowaniu definicji formalnej języka, ani że znajdą odbicie w formalnej definicji Ady. Bardziej dokładnie omówiono te zagadnienia w następnym punkcie.

UŻYTKOWNICY FORMALNEJ DEFINICJI ADY

Przy założeniu, że można opracować idealną definicję formalną Ady, obejmującą wszystkie pożądane właściwości, należy zapytać, kto będzie takiej definicji używać? Na podstawie rozważań z poprzedniego punktu, podjęto próbę określenia jej użytkowników.

Projektanci języka

Projektanci języka są zwykle ekspertami w kodowaniu programu (tzn. jego implementowaniu) i w implementowaniu języka (np. w pisaniu kompilatorów). Poza niewieloma wyjątkami [11] nie opanowali oni stylistycznie języka naturalnego (np. angielskiego) w stopniu wystarczającym do napisania precyzyjnego podręcznika języka programowania. Tym niemniej, projektanci jako jedyni — lub przynajmniej jako pierwsi — czytają taki nieformalny dokument. Obecnie są oni nieco lepiej przygotowani do czytania klasycznej postaci definicji formalnej [8].

Profesor DINES BJØRNER jest wykładowcą Politechniki Duńskiej w Lyngby i dyrektorem naukowym Dansk Datamatik Center. Jest on wybitnym teoretykiem i praktykiem informatyki, jednym z twórców tzw. nowej metody wiedeńskiej (VDM) specyfikacji i projektowania oprogramowania. W ubiegłym roku przebywał w Polsce na zaproszenie Polskiego Towarzystwa Informatycznego. Jest przewodniczącym Komitetu Programowego X Kongresu IFIP.

Mimo tych zastrzeżeń, definicja formalna powinna spełniać swoją podstawową rolę, tzn. wyjaśniać projektantom wszystkie wątpliwości dotyczące języka, tj. niejednoznaczności, niezdefiniowania, sprzeczności i niezupełność. Inna rola definicji formalnej wynika z samego podjęcia próby formalnego zdefiniowania języka. Łatwość lub trudność, z jaką postępuje proces definiowania, może stanowić pewną „miarę" naturalności lub sztuczności proponowanej konstrukcji językowej. Ten ostatni postulat nie jest jednak obiektywny w wypadku, gdy wybrana metoda definiowania (technika i język opisu semantyki) nie jest odpowiednia do tego celu.

Reasumując, projektanci języka, a także jego opiniodawcy (ang. distinguished reviewers) powinni być podstawowymi użytkownikami definicji formalnej Ady, pełniąc także rolę doradczą dla komitetów normalizacyjnych, w wypadku propozycji wprowadzenia zmian w języku.

Twórcy implementacji

Twórcy implementacji Ady (implementatorzy) mogą oczekiwać od definicji formalnej języka wyjaśnienia trzech rodzajów wątpliwości dotyczących:
- znaczenia odpowiednich właściwości języka, m.in. odpowiedzi na pytanie: co określona konstrukcja „ma robić",
- implementacji,
- zatwierdzania kompilatorów lub ich określonych składników.

Niekiedy uważa się, że są to raczej funkcje ośrodka atestowania (ang. validation centre), a nie definicji formalnej.

Z pewnością definicja formalna Ady powinna rozstrzygać pierwszy z wymienionych problemów. O ile sama definicja formalna jest określona konstruktywnie, np. przy użyciu formalizmu metody denotacyjnej lub operacyjnej, to może służyć również do wyjaśnień dotyczących drugiego punktu. Jeżeli definicja formalna może stanowić punkt odniesienia dla atestacji, to może być również użyteczna w punkcie trzecim. Sposób wykorzystania definicji formalnej do zatwierdzania programów (atestacji itp.) może polegać na użyciu jej:
- do dowodzenia poprawności implementacji na podstawie zgodności z definicją,
- do automatycznego generowania poprawnych i niepoprawnych programów służących do testowania implementacji.

Reasumując, twórcy kompilatorów, interpreterów i narzędzi wspomagających potrzebują definicji formalnej Ady do podejmowania decyzji dotyczących kwestii językowych.

Programiści

Programistów można podzielić na trzy grupy, ze względu na ich stosunek do jednostek rodzajowych:
- początkujących, którzy nigdy nie słyszeli o tych jednostkach,
- praktykujących, którzy ich używają,
- zaawansowanych, którzy je wytwarzają.

Niekiedy uważa się, że żadna z tych grup nie stanowi potencjalnych użytkowników formalnej definicji Ady. Jednakże należy sądzić, że inne grupy użytkowników, w szczególności wykładowcy (autorzy podręczników Ady) i lokalni eksperci (np. konsultanci) powinni działać jako pośrednicy między definicją formalną Ady a programistami.

Z drugiej strony, wskazuje się na bardzo pożyteczną rolę programistów, którzy znając formalną definicję Ady stwierdzają rozbieżności między zatwierdzonym kompilatorem a podręcznikiem języka, między dwoma zatwierdzonymi kompilatorami lub przyczyniają się do wyjaśnienia znaczenia elementów języka niezrozumiale opisanych w podręcznikach.

Członkowie komitetów normalizacyjnych

Członkowie organizacji normalizacyjnych (ISO, ECMA, ANSI, BSI, DOD) [1]) i komitetu pielęgnacji języka Ada (ang. Ada Language Maintenance Committee) działają na podstawie wskazówek otrzymanych z ośrodków atestowania (w celu ustalenia właściwej interpretacji), od projektantów i opiniodawców (w celu podjęcia decyzji o wprowadzeniu zmian do języka) i od twórców implementacji (np. w celu ułatwienia realizacji kompilatorów). W przeciwnym wypadku uwzględniają oni, na przykład, sugestie producentów, zmierzające do tworzenia podzbiorów lub rozszerzeń języka, sprzęgania go z językami systemów baz danych itp.

Uważa się, że członkowie tych komitetów powinni być najlepiej zaznajomieni z definicją formalną Ady i najbardziej zainteresowani jej pielęgnacją. Użycie definicji formalnej i jej kolejnych wersji powinno pomóc im w podjęciu decyzji dotyczących wprowadzenia proponowanych zmian w języku. Przed próbą wprowadzenia takiej zmiany do definicji formalnej Ady, można zastanowić się, czy ona jest pożądana. Bierze się przy tym pod uwagę łatwość jej wprowadzenia do definicji formalnej, skutki dla zakresu zmian w samym języku i wpływ na rozmiar samej definicji formalnej Ady.

Wykładowcy, nauczyciele i konsultanci

Do tej kategorii użytkowników zalicza się autorów książek, podręczników i poradników dotyczących Ady. Głównie w tym aspekcie rozumie się ich rolę w użyciu formalnej definicji języka.

Oczekuje się, że poświęcą oni czas na przestudiowanie formalnej definicji Ady i na tej podstawie opracują różnego rodzaju dokumenty opisujące w sposób nieformalny wybrane elementy języka. Definicja powinna pozwolić im na dokonanie wyboru określonych podzbiorów, aby umożliwić wprowadzenie studentów i programistów w zagadnienia języka oraz zróżnicowanie zajęć według specjalności słuchaczy. Ponadto, definicja formalna Ady powinna służyć im do rozstrzygania kwestii językowych, pojawiających się przy dokładniejszym wniknięciu w techniczne szczegóły języka, np. w wypadku wykrycia rozbieżności między zatwierdzonym kompilatorem a podręcznikiem itp.

Naukowcy

Przez naukowców rozumie się osoby zajmujące się teoretycznymi podstawami informatyki (ich formalizacją). Potencjalnie stanowią oni grupę użytkowników intensywnie wykorzystujących definicję formalną Ady. Definicji jako podstawy mogą używać do:

- wyprowadzania reguł dowodzenia poprawności programów w Adzie, przy założeniu, że sama definicja formalna nie jest sformułowana przy użyciu tych reguł,
- wyprowadzania reguł transformacji programów w Adzie, np. do stosowania w programowaniu lub w optymalizacji kompilatorów,
- wyprowadzania formalnego opisu semantyki języka specyfikacji do definiowania właściwości programów (jako rozszerzenie dwóch poprzednich punktów).

Ponadto, definicja formalna Ady może być przez nich używana jako punkt wyjścia:
- do badań dotyczących jej podstaw semantycznych, nie wymienionych powyżej, np. w dziedzinach nie objętych samą definicją, jak: ocena jakości działania, złożoność obliczeniowa, zagadnienia czasu rzeczywistego itp.,
- do badania innych, konkurencyjnych metod definiowania semantyki, np. wykazania, że są gorsze oraz wprowadzania udoskonaleń itp.

Opracowanie formalnej definicji Ady jest dużym przedsięwzięciem, prowadzącym do uzyskania bardzo obszernego dokumentu. Dokument, w którym próbuje się rozwiązać wszystkie możliwe kwestie, z pewnością nie będzie doskonały. Dlatego definicja formalna Ady musi być dokumentem żywym, poddanym ciągłej krytyce naukowców.

Ośrodki atestowania

Wzajemne odniesienia definicji formalnej Ady i ośrodków atestowania kompilatorów (jak też poprzednio omówionych grup użytkowników) stanowiły przedmiot prac organizacji Ada Europe. Poniższe uwagi pochodzą głównie z dokumentów roboczych tej organizacji.

Przez atestację (ang. validation, uwiarygodnienie) rozumie się na ogół proces, w którym biorą udział dostawca

[1]) ISO — International Organization for Standardization, ECMA — European Computer Manufacturers Association, ANSI — American National Standards Institute, BSI — British Standards Institution, DOD — Department of Defense (amerykański Departament Obrony).

i nabywca oprogramowania, mający na celu zwiększenie zaufania do poprawności specyfikacji, projektu lub implementacji (tj. kodu) lub stwierdzenie, że specyfikacja, projekt lub implementacja spełnia określone wymagania.

Na ogół, na proces ten składa się formalne dowodzenie poprawności i wykonanie testów (testowanie). Dowodzenie poprawności dotyczy kodu pod względem zgodności ze specyfikacją. Dowód (twierdzenia) może być wykonany automatycznie przez specjalny program (ang. theorem prover). Ręcznie wykonany dowód może być także zweryfikowany przez program (ang. proof checker). Twierdzenia są uzgadniane przez dostawcę i nabywcę. Podobnie, przedmiotem kontraktu dostawcy z nabywcą jest wybór testów i ich oczekiwanych wyników.

Testowanie stanowi kombinację dwóch czynności:
- systematycznego i zorganizowanego poszukiwania negatywnego przykładu na to, że specyfikacja, projekt lub implementacja jest poprawna,
- choćby częściowego wykonania specyfikacji, projektu lub kodu w celu wykazania, że spełnione są pewne wymagania niefunkcjonalne.

Dowody poprawności dotyczą zwykle tylko wymagań funkcjonalnych (poddających się formalizacji).

Na tym tle definicja formalna Ady w ośrodkach atestowania może służyć wielu celom:
- jako podstawa do zorganizowania systematycznego poszukiwania negatywnych przykładów dla dowodów poprawności,
- jako podstawa do generowania programów testowych, zarówno poprawnych, jak i niepoprawnych,
- jako podstawa do podjęcia decyzji dotyczących testów właściwości niefunkcjonalnych, tzn. nie objętych specyfikacją.

W odniesieniu do drugiego z powyższych punktów, można wskazać dwie możliwości użycia formalnej definicji Ady:
- do syntaktycznego generowania poprawnych i celowo niepoprawnych programów w Adzie,
- do generowania odpowiedzi kompilatora na wykonanie tak wygenerowanych programów.

Obecna wiedza na te tematy jest jednak jeszcze bardzo mała.

TECHNIKI SPECYFIKACJI JĘZYKA

Język może być zdefiniowany kilkoma sposobami: za pomocą podręcznika, przy użyciu przewodnika implementacyjnego, przez samą implementację lub matematycznie. Poniżej zajęto się tylko ostatnią postacią opisu, tj. definicją formalną.

Specyfikacje dedukcyjne i konstruktywne

Do opracowania definicji formalnej Ady wyróżniono dwa zasadniczo odmienne style formalizacji: dedukcyjny, zwany też asercyjnym (ukierunkowany na właściwości) i konstruktywny (ukierunkowany na modele).

Specyfikacje obejmujące systemy dowodzenia poprawności (ang. proof systems), tzn. specyfikacje prowadzące bezpośrednio do wnioskowania, są zazwyczaj dedukcyjne (asercyjne, ukierunkowane na właściwości). Takie specyfikacje wymagają zwykle dowodu swej niesprzeczności i zupełności oraz faktu, że coś definiują, tzn. że mają przynajmniej jeden model. Specyfikacje ukierunkowane na modele — jak wynika z nazwy — bezpośrednio opisują modele, tzn. są nimi same i w tym sensie są konstruktywne.

W praktyce zwykle żąda się określonych właściwości, lecz specyfikuje model. Na taką sytuację składa się kilka czynników:

- wykształcenie większości programistów jest ukierunkowane na modele
- obecne techniki specyfikacji ukierunkowanych na modele nadają się do definiowania znacznie bardziej złożonych systemów niż techniki dedukcyjne
- specyfikację trzeba zawsze doprowadzić do implementacji, tzn. prędzej lub później należy znaleźć model
- w chwili obecnej brakuje metod przechodzenia od definicji dedukcyjnej do konstruktywnej specyfikacji.

Definicja ukierunkowana na właściwości jest zbliżona do sposobu w jaki formułuje swoje wymagania nabywca, natomiast specyfikacja konstruktywna odpowiada sposobowi myślenia dostawcy, polegającemu na wyprowadzeniu implementacji ze specyfikacji.

Idealnie pożądane byłoby istnienie czystej definicji dedukcyjnej, np. Ady, umożliwiającej ścisłe wyprowadzenie konstruktywnej specyfikacji. W praktyce można oczekiwać, że jest możliwe dowiedzenie, iż teoria zdefiniowana dedukcyjnie (tj. aksjomaty definicji dedukcyjnych itd.) będzie spełniana przez model wyspecyfikowany konstruktywnie (tzn. będzie miała jego właściwości).

Jak można się domyślić na podstawie powyższych rozważań, punktem wyjścia obecnej definicji formalnej Ady jest ukierunkowanie na modele, lecz tak, aby wraz z rozwojem definicji zapewnić możliwość raczej bezpośredniego wyprowadzania właściwości.

Poniżej przedstawiono bardzo krótki przegląd technik specyfikacji, pokrywających cały zakres, od definicji dedukcyjnych do ukierunkowanych na modele, a więc semantykę aksjomatyczną, algebraiczną, operacyjną, denotacyjną i mechanistyczną.

Semantyka aksjomatyczna

Mówiąc niedokładnie, semantyka aksjomatyczna specyfikuje relacje między stanami specyfikowanego systemu. W aksjomatycznej specyfikacji języka programowania, jego semantyka jest określona przez aksjomaty i reguły dedukcyjne (korzystania z tych aksjomatów) [6].

Takie systemy aksjomatyczne wydają się idealne, jako systemy dowodzenia poprawności dla języka, który specyfikują. Jednakże w wypadku tak złożonego języka jak Ada, ich stosowanie może być bardzo uciążliwe, o ile w ogóle ma sens. Najistotniejszy wpływ na złożoność mają: instrukcje goto, procedury, przekazywanie parametrów i wielozadaniowość.

Tę dość negatywną opinię o przydatności semantyki aksjomatycznej jako podstawy formalnej definicji Ady, należy rozumieć właściwie. Posługując się tą metodą do specyfikacji semantyki można projektować wspaniałe języki i skutecznie ich używać, na co jest wiele przykładów w literaturze [7, 10]. Aksjomaty są wtedy wyrażone w nieco innym stylu i nazwane prawami. Jest bardzo prawdopodobne, że w przyszłości języki projektowane będą w oparciu o systemy dowodzenia poprawności, podlegające jednoczesnemu rozwojowi.

Semantyka algebraiczna

Semantyka algebraiczna specyfikuje znaczenie systemu jako klasę algebr. W algebraicznej specyfikacji języka programowania jego semantyka jest określona jako opis algebry, złożonej z sygnatury i zbioru aksjomatów. Zazwyczaj opis algebraiczny jest ograniczony syntaktycznie tak, aby zagwarantować istnienie modeli. Znaczeniem opisu algebry jest zwykle pewna klasa (kategoria) algebr. Aksjomaty są najczęściej określone równaniami. Najłatwiej dostępnym wprowadzeniem do semantyki algebraicznej jest [5].

Wydaje się, że algebraiczna specyfikacja semantyki powinna również nadawać się idealnie na podstawę systemu dowodzenia poprawności dla języka. Jednak — jak dotąd — nie powstała jeszcze algebraiczna specyfikacja języka o znaczących rozmiarach (o wielkości co najmniej Algolu 60), nawet bez uwzględniania współbieżności w tym języku. Problemy związane ze stosowalnością tej metody (oprócz wymienionych w poprzednim punkcie) polegają na jej niedostosowaniu do operowania funkcjami wyższych rzędów (procedurami o parametrach będących procedurami) i do wielozadaniowości.

Ta dość negatywna opinia o semantyce algebraicznej oznacza tylko tyle, że definicja Ady nie powinna być wyrażona bezpośrednio przy użyciu tej metody. Może się jednak okazać, że ostatecznie przyjęty sposób tworzenia definicji będzie wymagał algebraicznego określenia semantyki języka definicyjnego.

Semantyka denotacyjna

Semantyka denotacyjna określa znaczenie systemu w postaci zbioru obiektów matematycznych (jak dziedziny, funkcje, kategorie). W denotacyjnej specyfikacji języka programowania przede wszystkim identyfikuje się specyficzne obiekty metematyczne, które mają być przyłączone

do prostych identyfikatorów programów. Przykładowo, identyfikatory zmiennych mogą oznaczać (ang. denote) funkcje z tzw. środowisk na miejsca; identyfikatory etykiet mogą oznaczać tzw. kontynuacje, tzn. funkcje z pamięci na pamięć; identyfikatory procedur mogą oznaczać funkcje z list argumentów na kontynuacje itd. Następnie wyraża się znaczenie złożonych konstrukcji językowych jako funkcje znaczenia ich prostych składowych, co w rzeczywistości stanowi zasadę algebraiczną (homomorficzną) i nie charakteryzuje wyłącznie specyfikacji denotacyjnych.

Specyfikacje denotacyjne bezpośrednio specyfikują modele. Nie są one więc bezpośrednio użyteczne jako systemy dowodzenia poprawności. Jak dotychczas, nie prowadzono też wielu prac w tym kierunku i nie opracowano systematycznych technik wyprowadzania systemów dowodzenia ze specyfikacji denotacyjnych. Siła semantyki denotacyjnej polega na jej skuteczności w traktowaniu instrukcji goto, procedur, parametrów i wyjątków oraz większości innych deterministycznych cech języka. Metoda ta nie radzi sobie natomiast dobrze ze zmiennymi współdzielonymi, procesami podobnymi do występujących w Adzie i z indeterminizmem. Uwagi te dotyczą dostępności technik przeznaczonych do dużych zastosowań. Wyniki ostatnich badań (np. [1]) wydają się bardzo obiecujące, lecz dla tego projektu mają niestety jeszcze zbyt eksperymentalny charakter.

Strukturalna semantyka operacyjna

Strukturalna semantyka operacyjna (ang. Structural Operational Semantics, SOS) definiuje znaczenie systemu przez zbiór wszystkich dopuszczalnych sekwencji przejść, które występują w systemie, w czasie wykonania. W tym sensie (wykonawczym) semantyka SOS jest operacyjna.

Jej strukturalność polega na tym, że reguły przejść są wyspecyfikowane indukcyjnie, w oparciu o strukturę języka wejściowego systemu.

Specyfikacja SOS jest zwykle określana przez zbiór reguł przejść i służące do ich użycia reguły indukcyjne. Reguła przejścia składa się z trzech elementów: opisu stanu początkowego, opisu stanu końcowego i warunku (oznaczonego etykietą), pod którego wpływem system może przejść od konfiguracji początkowej do końcowej. Pojęcia konfiguracji i etykiety są dość luźno zdefiniowane i mogą obejmować takie składowe stanu, jak: pamięci fragmenty programu i inną informację sterującą.

Specyfikacje SOS doskonale modelują indeterministyczne i współbieżne elementy języka i wydają się obiecujące jako podstawa bezpośrednich i wyprowadzanych systemów dowodzenia poprawności. Zastosowane do pełnej Ady techniki specyfikacji SOS powodują powstanie dość złożonych konfiguracji i etykiet.

Inne techniki specyfikacji

Jak starano się wykazać, żadna z przedstawionych technik, być może z wyjątkiem SOS, nie jest całkowicie odpowiednia do specyfikacji pełnej Ady. Oprócz wymienionych stosuje się też inne techniki, jak np. SEMANOL [2], VDL [9] i META-IV/CSP [4]. Nie ma tu jednak miejsca na pobieżny nawet opis tych metod, zaliczanych do definicji operacyjnych (mechanistycznych).

Z przedstawionych, choć dość ogólnych uwag wynika, że pierwsza definicja formalna Ady nie będzie oparta w całości na żadnej z tych metod. Uważa się, że dla tak złożonego systemu — jak Ada, przede wszystkim istnieje potrzeba ustalenia modelu semantycznego. Znaczy to, że należy zapomnieć o dogmatach semantyki aksjomatycznej, algebraicznej lub denotacyjnej i ich opisie matematycznym, rozpoczynając od spraw podstawowych.

Tłumaczył i opracował
JANUSZ ZALEWSKI

LITERATURA

[1] de Bakker and Zuker J.: Processes and the Denotational Semantics of Concurrency. Information and Control, Vol. 54, pp. 70—120 (1982)

[2] Belz F. C., Blum E. K., Heimbinger D. M.: A Multi-Processing Implementation-Oriented Formal Definition of Ada in SEMANOL. SIGPLAN Notices, Vol. 15, No. 11, November 1980

[3] Bjørner D.: The Role and Scope of the Formal Definition of Ada. Report No. FD/DDC/01/V2.0, Dansk Datamatik Center, Lyngby, September 1985

[4] Bjørner D., Oest O. N.: Towards a Formal Description of Ada. Lectures Notes in Computer Science, Vol. 98. Springer Verlag, Heidelberg, 1980

[5] Ehrig H., Mahr B.: Algebraic Semantics — Initial Semantics and Equations. EATCS Series, Vol. 5. Springer Verlag, Heidelberg, 1985

[6] Hoare C. A. R.: The Axiomatic Basis of Computer Programming. Comm. of the ACM, Vol. 12, No. 10, pp. 567—583, October 1969

[7] Hoare C. A. R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs (NJ), 1985

[8] Hoare C. A. R., Wirth N.: An Axiomatic Definition of the Programming Language Pascal. Acta Informatica, Vol. 2, pp. 335—355 (1973)

[9] Lucas P. et al.: Method and Notation for the Formal Definition of Programming Languages. Report TR 25.087, IBM Laboratory, Vienna, July 1970

[10] Milner A. J. R.: Calculus of Communication Systems. Lecture Notes in Computer Science, Vol. 94. Springer Verlag, Heidelberg, 1980

[11] Naur P.: Revised Report on the Algorithmic Language Algol 60. Comm. of the ACM, Vol. 6, No. 1, p. 1 (1963).

WARUNKI PRENUMERATY NA 1986 R.

Prenumeratorzy zbiorowi — jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat na blankiecie „polecenie przelewu" rozszerzonym dla potrzeb Wydawnictwa o część dotyczącą zamówienia. Blankiety te będą dostarczane przez Zakład Kolportażu.

Prenumeratorzy indywidualni — osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie Wydawnictwa lub blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty.
Wpłacać należy na konto NBP III O/M Warszawa 1036-7490-139-11.

Prenumerata ulgowa — przysługuje wyłącznie osobom fizycznym — członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły.

Sposób zamawiania prenumeraty taki sam jak dla prenumeraty indywidualnej.

Prenumerata ze zleceniem wysyłki za granicę — zamawia się tak jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy. Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Przedpłaty na prenumeratę przyjmowane są w terminach:
— do 10 listopada na I kwartał, I półrocze i cały rok następny,
— do 28 lutego na II, III, IV kwartał i II półrocze,
— do 31 maja na III, IV kwartał i II półrocze,
— do 31 sierpnia na IV kwartał.

U w a g a!

Wpłaty na dwumiesięczniki przyjmowane są na okresy półroczne lub roczne.

Informacji o prenumeracie udziela — Zakład Kolportażu Wydawnictwa NOT-SIGMA, ul. Bartycka 20, 00-716 Warszawa, lub skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 w. 249, 293, 297, 299 oraz 40-35-89 i 40-30-86.

Egzemplarze archiwalne czasopism — można nabyć za gotówkę w Klubie Prasy Technicznej w Warszawie ul. Mazowiecka 12, tel. 27-43-65 oraz w. Dziale Handlowym Wydawnictwa ul. Bartycka 20 skr. poczt. 1004, 00-950 Warszawa, na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.
Cena miesięcznika INFORMATYKA została ustalona na 120 zł za numer (35 zł — cena ulgowa).

Cena prenumeraty wg cennika					
kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
360	105	720	210	1440	420

KRZYSZTOF PERYCZ
Instytut Łączności
Oddział Gdańsk

Mikrokomputerowy system operacyjny S1 (I)

Wśród eksploatowanych obecnie mikrokomputerowych systemów operacyjnych pojawił się niedawno (1984 r.) nowy system o nazwie S1, zwracający uwagę wszechstronnie przemyślaną koncepcją, wyjątkową funkcjonalnością oraz przenośnością i szerokim polem zastosowań.

System S1 obsługuje mikrokomputery z procesorami: 68000, Z80, 8080, 8085, 8086/88, 80186, 80286 oraz 16032. W systemie tym działają translatory najpopularniejszych języków programowania, jak: Pascal, Fortran 77, C, Basic, a zapowiedziane są: Cobol, Modula-2, PL/I, Ada, Lisp oraz Snobol. Dostępny jest kompilator języka SL, w którym napisany jest S1, a także asemblery i asemblery skrośne. Istnieje też szereg konwerterów programów źródłowych napisanych w rozmaitych realizacjach języków Pascal i Basic na standard S1.

Z cech szczególnie zwracających uwagę podkreślić należy, zapewnianą przez S1, wymienność plików danych z popularnymi systemami w standardach: CP/M, MP/M II, MS-DOS, UNIX, XENIX, p-System, FLEX, IBM 3741 i DEC Files-11 oraz kompatybilność — na poziomie programów źródłowych — z programami użytkowymi eksploatowanymi pod nadzorem systemu UNIX.

Obszerne fragmenty dokumentacji [2] oraz skróconego opisu porównawczego [3] prezentowane są za zgodą twórców systemu S1, firmy Multi Solutions Inc.

OPIS OGÓLNY

S1 jest uniwersalnym systemem operacyjnym dla mikrokomputerów. Jego cechami charakterystycznymi są:
- niezależność sprzętowa,
- modularność konstrukcji,
- realizacja w języku maszynowym (S1 jest kompilowany dla maszyny docelowej),
- dostępność funkcji systemowych dla programów użytkowych,
- szerokie możliwości pracy w sieciach komputerowych,
- działanie w czasie rzeczywistym,
- zgodność systemu plików z istniejącymi systemami operacyjnymi,
- zarządzanie oknami programowymi na monitorach ekranowych.

Komputer — pracujący pod nadzorem tego systemu — działa funkcjonalnie tak samo, niezależnie od tego, czy jest oparty na procesorze Z80, 68000, czy jakimkolwiek innym 8-, 16- lub 32-bitowym. Zarówno język poleceń, jak i struktura plików są we wszystkich wypadkach identyczne. Identyczny jest również sposób korzystania z wywołań systemowych. Dzięki temu programy użytkowe mogą być przenoszone z procesora na procesor jedynie przez ponowne kompilowanie ich postaci źródłowej. Sam system S1 jest napisany w języku „średniego poziomu" SL w taki sposób, aby był niezależny sprzętowo. Fragmenty zależne od konfiguracji użytego sprzętu są zgrupowane w jednym krótkim segmencie.

Moduły wchodzące w skład systemu S1 nie są ze sobą ściśle powiązane, dzięki czemu mogą być łatwo usunięte lub dodane w miarę potrzeby. Zażądanie funkcji realizowanej przez usunięty moduł nie powoduje upadku systemu, lecz tylko sygnalizację „funkcja niedostępna". S1 nie jest wykonywany interpretacyjnie, dlatego działa z pełną szybkością komputera. Ponieważ system S1 jest zaprojektowany jako uniwersalny system operacyjny, możliwe jest dopasowanie go do pełnienia specyficznych zadań. Można używać go jako:
- systemu dla urządzenia do zbierania danych,
- systemu do sterowania w czasie rzeczywistym,
- systemu operacyjnego dla komputera osobistego,
- wielodostępnego i wielozadaniowego systemu z podziałem czasu,
- systemu operacyjnego dla sieci rozproszonych,
- systemu wspomagania.

Programy użytkowe korzystają ze wszystkich możliwości systemu. Jakakolwiek jego funkcja, jak np. przydzielenie bloku z pamięci dyskowej lub dołączenie terminala do zadania, może być wywołane z programu. Ułatwia to rozszerzenie i utrzymanie systemu.

System S1 może zarządzać grupami wzajemnie połączonych komputerów. W takim wypadku pliki i urządzenia są na ogół wspólnie używane. Fakt użycia zasobu innego komputera może być niezauważalny dla programu użytkowego. Dostępne są także funkcje elektronicznej poczty.

Realizacji sterowania w czasie rzeczywistym służą takie cechy, jak:
- wielozadaniowość,
- dynamiczne przydzielanie procesora zadaniom,
- mechanizmy synchronizacyjne i zapewniające wyłączność korzystania z określonych zasobów systemu (semafory, blokady i komunikaty),
- zagwarantowanie szybkiego dostępu do informacji w plikach dyskowych (metodą przydzielania ciągłego obszaru dyskowego).

W systemie S1 możliwy jest odczyt i zapis plików w formatach stosowanych przez inne systemy operacyjne, zarówno na dyskach, jak i na taśmach magnetycznych. Umożliwia to bezpośrednią wymianę danych między tymi systemami.

Z zestawu funkcji zarządzających informacjami na ekranie monitora mogą korzystać nie tylko edytory czy programy kalkulacji arkuszowych, ale wszystkie programy użytkowe. Funkcje zarządzające oknami programowymi umożliwiają jednoczesną obserwację wyników pracy wielu programów. Informacje wyświetlane przez określony program nie kolidują z informacjami przesyłanymi na ekran przez inne programy, dzięki czemu można w przejrzysty sposób zawiesić wykonywanie jednego programu, wykonać inny program, a następnie kontynuować ten zawieszony. Jest to szczególnie cenna cecha w wypadku testowania i eksploatacji oprogramowania graficznego.

PROCESORY POLECEŃ

Użytkownik zleca zadania do wykonania systemowi za pośrednictwem języka poleceń. W skład systemu wchodzą trzy podstawowe procesory poleceń:

- procesor konwencjonalny, który przyjmuje z terminala linię polecenia, analizuje ją, określa program realizujący to polecenie i przekazuje mu sterowanie; ten rodzaj procesora jest używany najczęściej w mikrokomputerowych systemach wspomagania, w systemach minikomputerowych i w dużych maszynach cyfrowych;

- procesor typu „menu" wyświetlający kolejno listy możliwych poleceń; wyboru polecenia dokonuje się, zależnie od złożoności terminala, przez napisanie numeru, przesunięcie kursora na odpowiednią pozycję klawiszami lub dodatkowym urządzeniem bądź przez dotknięcie piórem świetlnym; użytkowanie takiego procesora poleceń jest znacznie łatwiejsze, lecz dla doświadczonego operatora jest on zbyt wolny;

- procesor „podpowiadający", który różni się od konwencjonalnego tym, że niepodanie argumentu polecenia nie

powoduje wydania komunikatu o błędzie i zarzucenia wykonania polecenia, lecz wyświetlenie żądania stosownego argumentu; procesor taki stanowi pewien kompromis między poprzednimi dwoma rodzajami i pozwala na użytkowanie systemu osobie, która nie pamięta szczegółowo składni wszystkich poleceń; jest on szczególnie skuteczny przy stosowaniu terminali z programowanymi klawiszami funkcyjnymi.

POLECENIA SYSTEMU S1

Do najważniejszych poleceń systemu S1 należą następujące:

assemble — translacja pliku źródłowego z programem w języku asemblera na plik wynikowy w formacie oczekiwanym przez program łączący,

backup — kopiowanie archiwalne wybranych plików, tj. całej zawartości dyskietki lub dysku, bądź też tych tylko plików, które spełniają pewne warunki (np. zostały zmodyfikowane od czasu wykonania poprzedniej operacji backup),

calculate — użycie systemu S1 jako kalkulatora; wykonywane są operacje dodawania, odejmowania, mnożenia i dzielenia liczb w systemie ósemkowym, dziesiętnym lub szesnastkowym (liczby dziesiętne mogą zawierać część ułamkową, podaną po kropce),

charge — utworzenie pliku zawierającego dane o zużyciu zasobów przez użytkowników systemu,

clean — kasowanie wszystkich plików, które zostały usunięte ze skorowidza; operacja ta jest wykonywana dla jednego wyspecyfikowanego urządzenia, choć można zażądać jej przeprowadzenia dla wszystkich urządzeń,

compare — porównanie, czy dwa pliki są identyczne; jeśli się różnią, to generowana jest lista różnic, która może być sformatowana znakowo, szesnastkowo lub ósemkowo; jeśli plik jest sekwencją linii, to porównanie wykrywa linie wstawione i usunięte,

connect — włączenie systemu do sieci (w systemach wielodostępnych jest to operacja uprzywilejowana),

copy — kopiowanie pliku (oprócz kopiowania zawartości zwykłych plików można także kopiować skorowidze, jak również zawartość wielu plików do jednego pliku),

debug — wykonywanie standardowych funkcji testowania i uruchamiania programu, tj. wyświetlania i zadawania zawartości komórek pamięci, punktów wstrzymania oraz informacji sterujących; polecenie to umożliwia również śledzenie zawartości zmiennych, co pozwala na wykrywanie przyjęcia przez wybraną zmienną zadanej wartości,

decrypt — zdekodowanie zawartości zaszyfrowanego pliku na jego jawną postać (odwraca skutki działania polecenia encrypt),

destroy — całkowite zniszczenie pliku; polecenie to jest używane do eliminowania plików tymczasowych lub takich, które są z całą pewnością zbędne,

disconnect — odłączenie systemu od sieci (w systemach wielodostępnych jest to operacja uprzywilejowana),

display — wyświetlenie informacji systemowych należących do klasy SYSTEM VARIABLES,

edit — przeglądanie lub modyfikacja zawartości pliku; w systemach z monitorami ekranowymi edycja może być wykonywana w trybie pełnoekranowym,

encrypt — zaszyfrowanie zawartości pliku na postać nieczytelną bez znajomości klucza; szyfrowanie może odbywać się wielopoziomowo, przy czym każdy następny szyfr jest trudniejszy do złamania, lecz wymaga więcej obliczeń do zakodowania i zdekodowania,

erase — usunięcie pliku ze skorowidza; zawartość pliku jest utrzymana na dysku aż do jego zapełnienia (por. polecenia clean, destroy i unerase),

export — zapisanie danych na nośnikach w formacie i kodzie zgodnym z systemem, dla którego mają być przeznaczone,

help — wyprowadzenie informacji dotyczących użytkowania systemu, wyjaśnienie komunikatów o błędach oraz poleceń,

import — odczytywanie danych z nośników w formacie i kodzie obcych systemów komputerowych,

initialize — przygotowanie nośników do eksploatacji w systemie S1 (np. w wypadku dyskietek formatuje się je oraz wypełnia potrzebnymi informacjami sterującymi),

link — utworzenie pliku z programem wykonywalnym przez połączenie plików wynikowych; zazwyczaj pliki wynikowe są plikami wyjściowymi kompilatorów i asemblerów,

list — wyprowadzenie na terminal nazw plików; argumenty polecenia definiują zakres i rodzaj wyprowadzanych informacji,

logoff — wyrejestrowanie użytkownika z systemu,

logon — zarejestrowanie użytkownika, co umożliwia systemowi operacyjnemu jego identyfikację,

mail — przesyłanie komunikatów i plików do innych użytkowników (także w innych punktach sieci); cała „poczta" stanowi kolejkę i można nią manipulować dostępnymi funkcjami działającymi na kolejkach, można np. zażądać poczty wysłanej po pewnej dacie,

order — zmiana kolejności komunikatów w kolejce,

record — rejestrowanie informacji pojawiającej się na terminalu; normalnie stosuje się rejestrację dla terminali niepełnoekranowych,

relogon — przerejestrowanie, tj. przejście do pracy z innym identyfikatorem,

remotelogon — identyfikacja użytkownika przez inny system w sieci i rozpoczęcie pracy w tym systemie,

rename — zmiana nazwy pliku lub nazwy grupy plików,

repair — dostęp do uszkodzonych danych w celu ich odzyskania w wypadku stwierdzenia błędów uniemożliwiających odczytanie informacji z pliku konwencjonalnym systemem we-wy (często jest możliwe poprawne odczytanie wszystkich plików mimo całkiem nieczytelnego skorowidza),

set — zmiana informacji systemowych; pewne zmienne systemowe nie mogą być zmieniane bezpośrednio (np. wielkość wolnej przestrzeni na dysku); w wersjach wielodostępnych nieuprzywilejowanym użytkownikom nie pozwala się zmieniać niektórych informacji systemowych,

sort — porządkowanie obiektów pliku w określonej kolejności; działa zarówno na pliki strumieniowe (jeśli są w nich znaki separujące linie), jak i rekordowe; linie pliku mogą być sortowane w porządku wzrastającym lub malejącym, a klucz do sortowania może być zdefiniowany jako pole zawarte między określonymi ogranicznikami lub jako pole o ustalonej pozycji w linii,

talk — natychmiastowa wymiana komunikatów między użytkownikami dołączonymi lokalnie bądź zdalnie,

transfer — przenoszenie komunikatów z jednej kolejki do innej; polecenie to może być używane do odzyskiwania treści komunikatów wysłanych do innych użytkowników,

type — wyprowadzenie zawartości pliku na terminal w formacie znakowym, szenastkowym lub ósemkowym,

unerase — przywrócenie dostępu do plików usuniętych ze skorowidza, lecz jeszcze nie skasowanych (np. komendą clean).

Oprócz wymienionych poleceń w systemie istnieją też programy przetwarzania tekstów, kalkulacji arkuszowych, kompilatory i interpretery różnych języków programowania.

LISTY POLECEŃ

W celu ułatwienia pracy w systemie S1 oraz zautomatyzowania przetwarzania, polecenia mogą być zorganizowane w listę, przechowywaną w pliku. Na ogół kolejne polecenie jest wywoływane po zakończeniu aktualnie wykonywanego; język procesora poleceń umożliwia jednak sterowanie kolejnością ich wykonywania.

Najważniejsze cechy języka poleceń są następujące:

- lista poleceń może mieć zmienne lokalne i globalne,
- polecenia mogą być wykonywane warunkowo, po wystąpieniu pewnych zdarzeń w trakcie realizacji listy poleceń,
- sekwencje poleceń mogą być powtarzane w pętli,
- błędy sygnalizowane przez polecenia mogą być obsługiwane w liście poleceń,

● lista poleceń umożliwia przesyłanie danych między plikami i terminalami w obu kierunkach.

Listy poleceń stanowią efektywny sposób sterowania procesem ich wykonywania, tworząc swoisty program w prostym języku programowania. Zmiana w programie wymaga jedynie jego edycji. Dzięki temu można wygodnie testować nowe polecenie, a gdy okaże się ono użyteczne i często używane — można je wyrazić w dowolnym języku programowania.

LITERATURA

[1] Little J.: A Module Approach to Microcomputer Operating Systems, Computer Design, pp. 217—224, July 1984

[2] Multi Solutions Inc.: S1 Document S1-001-003, Lawrenceville, New Jersey, 1984

[3] Multi Solutions Inc.: S1 Highlights, Lawrenceville, New Jersey, 1984.

KRZYSZTOF MICHALIK
Instytut Organizacji Przetwarzania Danych
Akademia Ekonomiczna
Katowice

Programowanie logiczne

Wybór rozszerzonej wersji Prologu jako języka maszynowego wysokiego poziomu (ang. Kernel Language) komputerów tzw. piątej generacji spowodował znaczny wzrost zainteresowania programowaniem logicznym. Uznano tym samym znaczenie programowania logicznego między innymi w takich dyscyplinach jak: sztuczna inteligencja, bazy danych i inżynieria oprogramowania.

W artykule omówiono teoretyczne podstawy programowania logicznego, bez uwzględnienia konkretnego języka programowania. Pominięto również problem strategii sterowania, koncentrując się na składniku logicznych programów.

KLAUZULE HORNA

Logika zajmuje się m.in. badaniem relacji implikacji pomiędzy założeniami a wnioskami. Pozwala ona, na przykład, z następujących założeń:

Jan lubi programowanie logiczne **oraz**
Jan lubi każdego, kto lubi programowanie logiczne
wyprowadzić wniosek, że
Jan lubi siebie.

Aby wykazać, że dane założenia implikują określone wnioski, konieczne jest przeprowadzenie dowodu składającego się z kroków wnioskowania. Wymaga to precyzyjnego i jednoznacznego formułowania zadań logicznych.

Szczególnie interesująca w kontekście programowania logicznego jest klauzulowa postać logiki (ang. clausal form of logic). Najprostszymi zadaniami są tu tzw. atomy, przedstawiające określone relacje pomiędzy obiektami, np.:
Jan **lubi** programowanie logiczne.

Mgr KRZYSZTOF MICHALIK od 1981 r. pracuje w Zakładzie Informatyki Instytutu Organizacji Przetwarzania Danych Akademii Ekonomicznej w Katowicach na stanowisku starszego asystenta. Zajmuje się zagadnieniami programowania logicznego oraz systemów ze sztuczną inteligencją. Ponadto interesuje się zagadnieniami inżynierii oprogramowania.

Wyraz wytłuszczony jest nazwą relacji. Bardziej złożone zadania pozwalają wyrazić implikację:

Jan **lubi** x, jeśli x **lubi** programowanie logiczne.

W dalszej części artykułu wykorzystywana będzie notacja prefiksowa, w której nazwa relacji poprzedza obiekty, których dotyczy, np.:

Lubi(Jan, programowanie logiczne).

W tym miejscu niezbędne staje się podanie bardziej precyzyjnej definicji składni klauzul. Klauzula jest wyrażeniem o następującej postaci:

$B_1, ..., B_m \leftarrow A_1, ..., A_n$
gdzie $B_1, ..., B_m$ i $A_1, ..., A_n$ są atomami, oraz $m \geqslant 0$ i $n \geqslant 0$ [11].

W tym zapisie przecinki w części warunkowej klauzuli zastępują spójnik "i", natomiast w następniku implikacji — spójnik "lub". Jeśli klauzula zawiera zmienne $x_1,...,x_k$, to można ją interpretować:

dla każdego $x_1,...,x_k$
B_1 lub ... lub B_m, jeśli A_1 i ... i A_n.

Jeśli $n = 0$, to klauzulę można traktować jako stwierdzenie bezwarunkowe (aserację). Natomiast, gdy $m = 0$, to klauzulę o tej postaci nazywa się celem i interpretuje w następujący sposób:

dla każdego $x_1,...,x_k$
nieprawdą jest, że A_1 i ... i A_n.

Jeśli $m = n = 0$, to klauzulę taką nazywa się klauzulą pustą i interpretuje również jako sprzeczność. Klauzule puste będą oznaczone symbolem □.

Atomy są wyrażeniami o postaci: $P(t_1,...,t_m)$, gdzie P jest symbolem predykatu, a $t_1,...,t_m$ są termami, oraz $m \geqslant 1$. Termy są zmiennymi, stałymi lub wyrażeniami o postaci: $f(t_1,...,t_m)$, gdzie f jest symbolem funkcji, $t_1,...,t_m$ są termami oraz $m \geqslant 1$ [11].

W celu zwrócenia uwagi na wnioski, symbol implikacji zapisuje się często w pozycji odwróconej względem tradycyjnie przyjętej: $B \leftarrow A$ (B jeśli A).

Zmienne są oznaczone małymi literami. Należy podkreślić, że opisy obiektów w klauzuli mają charakter lokalny dla tej klauzuli.

Zgodnie z podaną składnią, stwierdzenie, że każdy człowiek jest mężczyzną lub kobietą (w sensie alternatywy wykluczającej) można zapisać za pomocą dwóch klauzul:
Kobieta(x), Mężczyzna(x) ← Człowiek(x).
← Kobieta(x), Mężczyzna(x), Człowiek(x).

Zdanie „Jan lubi x, jeśli x lubi programowanie logiczne" można teraz przedstawić następująco:

Lubi(Jan,x) ← Lubi(x,programowanie logiczne).

Do wielu zastosowań wystarczy bardziej ograniczona postać klauzul, mająca co najwyżej jeden wniosek (predykat) w następniku implikacji. Ta postać klauzuli nosi nazwę klauzul Horna [6]. Klauzule te odgrywają szczególną rolę w programowaniu logicznym, jak również w rozwiązywaniu problemów. Przykładem klauzul Horna są:

Mężczyzna(x) ← Ojciec(x,y)
Kobieta(x) ← Matka(x,y).

Bardziej formalne wyłożenie logicznych podstaw programowania logicznego można znaleźć w artykule [2].

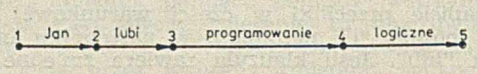
WNIOSKOWANIE I ROZWIĄZYWANIE PROBLEMÓW

W programowaniu logicznym wykorzystuje się głównie dwie metody wnioskowania: wnioskowanie zstępujące — przez analizę celu (ang. top-down) oraz wnioskowanie wstępujące — przez syntezę nowej informacji ze starej (ang. bottom-up). Obie te metody można traktować jako szczególne przypadki zasady rezolucji wprowadzonej przez J. A. Robinsona [13]. Wspomniane metody można zilustrować bardzo prostym problemem, dotyczącym analizy składniowej.

Zakłada się, że istnieją następujące reguły gramatyczne:

⟨zdanie⟩ : := ⟨podmiot⟩ ⟨orzeczenie⟩
⟨podmiot⟩ : := ⟨rzeczownik⟩
⟨orzeczenie⟩ : := ⟨czasownik⟩ ⟨dopełnienie⟩
⟨dopełnienie⟩ : := ⟨rzeczownik⟩ ⟨przymiotnik⟩
⟨rzeczownik⟩ : := Jan | programowanie
⟨czasownik⟩ : := lubi
⟨przymiotnik⟩ : := logiczne

oraz ciąg słów: Jan lubi programowanie logiczne. Na podstawie podanych założeń należy wykazać, że ten ciąg słów jest zdaniem. W tym celu przedstawione założenia doprowadza się do postaci klauzul Horna. Wejściowy ciąg słów można traktować jako graf o węzłach znajdujących się pomiędzy sąsiadującymi słowami. Słowa są nazwami łuków, łączących sąsiadujące węzły grafu:



Reguły podanej gramatyki traktuje się jako stwierdzenia odnoszące się do węzłów grafu (por. [11]). Dla przykładu, stwierdzenie:

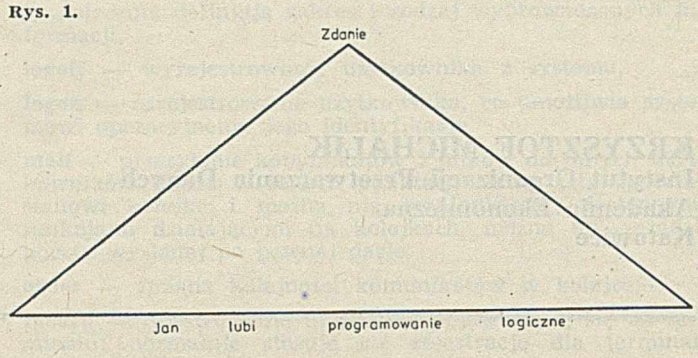
Prz(x,y) ← logiczne(x,y)
można interpretować w następujący sposób: jeśli istnieje droga z x do y nazwana "logicznie", to droga ta jest również nazwana "Prz". W rezultacie powstaje ciąg klauzul opisujących przedstawiony problem:

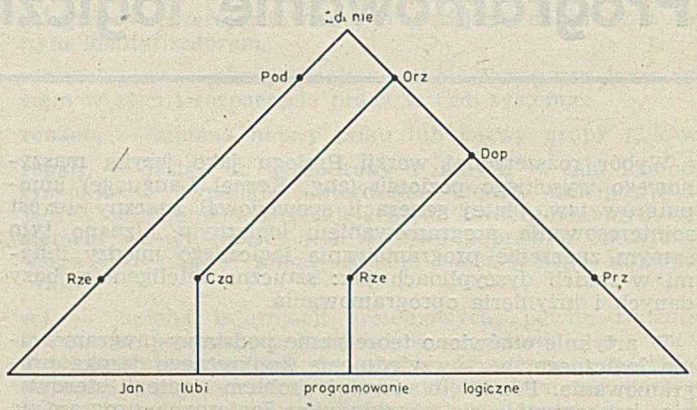
1. Jan(1,2) ←
2. lubi(2,3) ←
3. programowanie(3,4) ←
4. logiczne(4,5) ←
5. Zda(x,y) ← Pod(x,z), Orz(z,y)
6. Pod(x,y) ← Rze(x,y)
7. Orz(x,y) ← Cza(x,z), Dop(z,y)
8. Dop(x,y) ← Rze(x,z), Prz(z,y)
9. Rze(x,y) ← Jan(x,y)
10. Rze(x,y) ← programowanie(x,y)
11. Cza(x,y) ← lubi(x,y)
12. Prz(x,y) ← logiczne(x,y)
13. ← Zda(1,5)

Aby dowieść, że podany ciąg słów jest zdaniem w sensie przytoczonej gramatyki, co można zapisać jako Zda(1,5) ←, wystarczy wykazać, że negacja tego stwierdzenia (klauzula 13) jest niezgodna ze zbiorem klauzul 1—12. W praktyce oznacza to obalenie prawdziwości tej negacji. Pojawienie się niezgodności kończy wnioskowanie, co bardziej szczegółowo określono w dalszej części artykułu.

Rozwiązanie zaprezentowanego problemu można uzyskać, uzupełniając trójkąt o postaci przedstawionej na rys. 1, za pomocą wewnętrznie zgodnego drzewa rozbioru, tak by ten trójkąt przybrał ostatecznie postać z rys. 2.

Rys. 1.

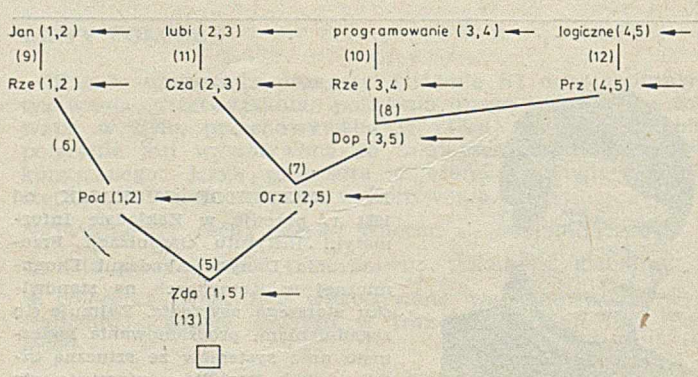




Rys. 2. Ilustracja rozbioru zdania z rys. 1

Metodę tę opisał S. Amarel [1]. Istnieją dwa podstawowe sposoby wypełniania tego trójkąta, odpowiadające wnioskowaniu wstępującemu oraz zstępującemu. Wnioskowanie wstępujące (syntetyczne) rozpoczyna się od aseracji zbioru wejściowego klauzul. Następuje tu uzgadnianie asercji z warunkami klauzul (predykatami w części warunkowej klauzul) i w rezultacie tworzone są nowe asercje. Uzgodnienie polega na ustaleniu identyczności nazw oraz argumentów wybranych predykatów dwóch klauzul lub na spowodowaniu tej identyczności przez dokonanie odpowiednich podstawień za zmienne. Dokładniejsze omówienie uzgodnienia zawarte jest w części artykułu dotyczącej programów logicznych.

Wnioskowanie kończy się w momencie uzyskania asercji stojących w sprzeczności z celem. Ten rodzaj wnioskowania można traktować jako uogólnienie reguły **modus ponens**, która orzeka, że jeśli A i B ← A, to B. Rozwiązanie przedstawionego wcześniej problemu, za pomocą wnioskowania wstępującego, można przedstawić następująco:



W powyższej reprezentacji rozwiązania linie łączą stare asercje z nowymi. Liczby w nawiasach, które są umie-

szczone przy tych liniach, oznaczają numery klauzul zastosowanych we wnioskowaniu. Dla przykładu, asercja:

Orz(2,5) ←
jest uzyskana na podstawie asercji:

A1: Cza(2,3) ←
A2: Dop(3,5) ←

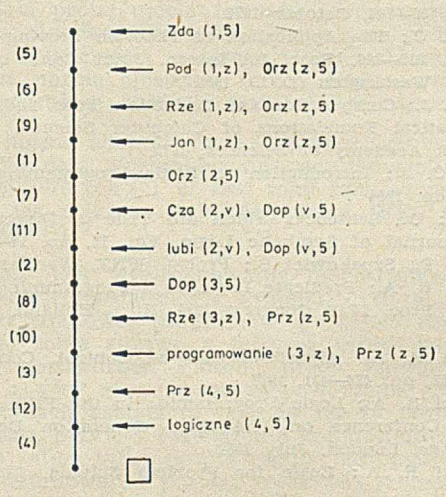
po zastosowaniu w kroku wnioskowania klauzuli 7 o postaci:

Orz(x,y) ← Cza(x,z), Dop(z,y).

Uzgodnienie warunków tej klauzuli z asercjami A1 i A2 nastąpiło po zastosowaniu podstawienia {x=2, z=3, y=5}. Klauzula pusta została wyprowadzona w rezultacie uzgodnienia asercji Zda(1,5) ←

z klauzulą 13. Ponieważ klauzula ta jest negacją (bez wniosków), to wyprowadzona klauzula jest klauzulą pustą.

Ten sam problem można rozwiązać za pomocą wnioskowania zstępującego. W tym wypadku procedura wnioskowania rozpoczyna się od zaprzeczenia jednej ze zbioru klauzul. Następnie wykorzystuje się implikacje oraz asercje, w celu uzyskania nowych zaprzeczeń. Wnioskowanie kończy się w momencie wyprowadzenia pustej klauzuli. Z punktu widzenia logiki ten typ wnioskowania jest związany z regułą **modus tollens**, na mocy której, jeśli nie B i B ← A, to nie A. Tym samym podany problem można rozwiązać w następujący sposób:



Liczby w nawiasach z lewej strony oznaczają numery klauzul zastosowanych we wnioskowaniu. W podanym przykładzie rozwiązania wnioskowanie rozpoczyna się od uzgodnienia klauzuli 13 z następnikiem implikacji w klauzuli 5. W rezultacie wyprowadzono nową klauzulę o postaci:

← Pod(1,z), Orz(z,5)

W tym wypadku zastosowano podstawienie {x=1, y=5}.

Jeśli klauzula zawiera kilka warunków, to do uzgodnienia wybiera się jeden z nich. Sposób wybierania warunków w klauzuli jest określony przez przyjętą strategię sterowania. W przedstawionym przykładzie warunki pobierane są kolejno od strony lewej do prawej. Wybrany warunek zastępuje się przez poprzedniki klauzuli zastosowanej w danym kroku wnioskowania. Jeśli ta klauzula jest asercją, to wybrany warunek jest usuwany. Jeśli ponadto warunek ten jest jedynym warunkiem w zaprzeczeniu (celu), to wyprowadzona zostanie klauzula pusta.

PROGRAMY LOGICZNE

Jedną z pierwszych prac dotyczących związku pomiędzy zasadą rezolucji a programowaniem komputerów był artykuł C. C. Greena [3]. Ideę traktowania zadań logicznych jako instrukcji programu rozważał m.in. P. J. Hayes [4]. Jednakże szczególny wkład do traktowania logiki jako języka programowania wniósł R. A. Kowalski [8].

Idea programowania logicznego opiera się na założeniu reprezentowania algorytmów w postaci dwóch rozdzielonych składników — logiki oraz sterowania, co Kowalski [9] poglądowo ujął jako:

algorytm = logika + sterowanie.

Powszechnie funkcjonuje inny schemat, odnoszący się do programów:

program = algorytm + struktury danych.

Przedstawiając za „algorytm" składniki „logika + sterowanie" z poprzedniego schematu, otrzymamy:
program = logika + sterowanie + struktury danych.

Ponieważ procedury logiczne mogą być traktowane same w sobie jako struktury danych, a sterowanie jest funkcją interpretera programu logicznego, zatem schemat dla programów logicznych przyjmuje ostatecznie postać (por. [5]):
program = logika.

W programach logicznych implikacje oraz asercje interpretowane są jako procedury, w których nagłówkami są następniki implikacji bądź asercje, a poprzedniki implikacji są traktowane jako treść procedur. Program logiczny składa się z celu oraz dowolnej liczby procedur. Cel o postaci: $?A_1,A_2,...,A_n$ może być interpretowany jako zbiór wywołań procedur o nazwach $A_1,...,A_n$. Argumenty predykatów są tu nazywane parametrami aktualnymi, natomiast argumenty predykatów będących nagłówkami procedur są nazywane parametrami formalnymi.

W celu zwiększenia czytelności przykładów w dalszej części artykułu zmieniono nieznacznie notację, wprowadzając zamiast znaku implikacji słowo „**if**", natomiast dla oznaczenia zaprzeczenia użyto znaku „**?**". W związku z tym przykładowy program logiczny mógł mieć następującą postać:

cel
procedury
C1: ? A, B
P1: B
P2: C
P3: A **if** B, C
gdzie A, B i C symbolizują predykaty.

Kolejny przykład programu logicznego dotyczy następstwa elementów danej listy. W podanej wersji pozwala on ustalić, jaki element następuje po liczbie 2 w liście o postaci:
1.2.3.4.NIL
gdzie kropki oznaczają funktor infiksowy (z łączeniem w prawo), natomiast stała NIL symbolizuje listę pustą. W tym zapisie
x.y
oznacza listę, której pierwszym elementem (głową) jest x, po którym następuje lista (ogon) y. W programie użyto predykatu nast(x,y,z), który oznacza, że element y następuje po x w liście z.
C1: ? **nast**(2,v,1.2.3.4.NIL)
P1: **nast**(x,y,x.y.z)
P2: **nast**(x,y,w.z) **if nast**(x,y,z)
Wykonanie tego programu za pomocą wnioskowania zstępującego spowoduje wygenerowanie następującego ciągu zaprzeczeń:
Z1: ? **nast**(2,v,1.2.3.4.NIL)
Z2: ? **nast**(2,v,2.3.4.NIL)
Z3: □

Każdy krok wnioskowania podczas wykonywania programu można traktować jako operację wywołania procedury. Zaprzeczenie Z2 powstało jako rezultat wywołania procedury P2 przez C1. Zaprzeczenie Z3 powstało wskutek wywołania P1 przez Z2.

Na operację wywołania procedury składają się następujące elementy:

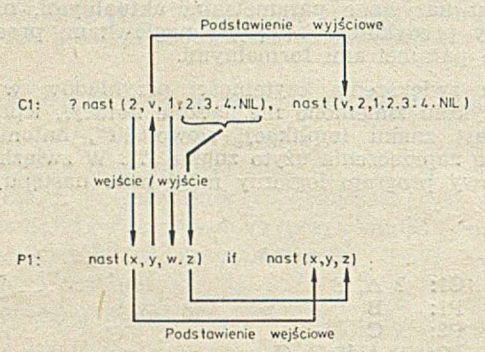
— wybór predykatu (wywołania procedury) w celu
— wybór procedury o nazwie zgodnej z nazwą predykatu w wywołaniu
— uzgodnienie parametrów formalnych i aktualnych
— zamiana wywołania w celu przez treść wywołanej procedury
— podstawienie przekazanych parametrów do nowo utworzonego celu.

Osobnym zagadnieniem jest mechanizm przekazywania parametrów procedur, który często określa się jako uzgodnienie lub unifikację (por. [7]). W celu wyjaśnienia tego zagadnienia podano za R. A. Kowalskim [11] definicję podstawienia, uzgodnienia oraz wcielenia.

Podstawienie $x_1=t_1,...,x_m=t_m$ jest zbiorem elementów o postaci $x_i=t_i$, gdzie x_i jest zmienną, a t_i jest termem. Jeśli W jest wyrażeniem (term, atom lub klauzula), to rezultatem zastosowania podstawienia: $\Theta=\{x_1=t_1,...,x_m=t_m\}$ do wyrażenia W, jest nowe wyrażenie $W\Theta$. Wyrażenie to jest identyczne z W, z wyjątkiem wystąpień x_i w wyrażeniu W, w miejsce których w $W\Theta$ występują teraz t_i. Wyrażenie $W\Theta$ jest również nazywane **wcieleniem** (ang. instance) wyrażenia W (por. [7]). Mówi się, że podstawienie σ **uzgadnia** dwa wyrażenia W_1 oraz W_2, wtedy gdy powoduje ono, że wyrażenia te stają się identyczne, tzn. $W_1\sigma=W_2\sigma$. W celu zilustrowania powiązań zmiennych w czasie uzgadniania podano przykład programu o postaci:

C1: ? **nast**(2,v,1.2.3.4.NIL), **nast**(v,2,1.2.3.4.NIL)
P1: **nast**(x,y,w.z) **if nast**(x,y,z)

Po wywołaniu procedury P1 przez C1 każda dana wejściowa przypisana określonej zmiennej x_i jest podstawiana w miejscu wszystkich wystąpień tej zmiennej w tekście procedury. Jednocześnie dowolne informacje wyjściowe przypisane określonej zmiennej y_i są podstawiane do wszystkich wywołań procedur (w części warunkowej klauzul), oczekujących na uaktywnienie. Można to zilustrować następującym schematem (por. [5]):



Należy podkreślić, że rozróżnienie pomiędzy parametrami wejściowymi i wyjściowymi procedur zależy wyłącznie od kontekstu, w jakim te procedury są wywoływane. Jest to bardzo ważna cecha programów logicznych. W praktyce oznacza to, że ta sama procedura może być wykorzystywana do realizacji różnych funkcji. Załóżmy, że istnieje procedura dodawania liczb, wywoływania za pomocą predykatu o postaci:

Dodaj(x,y,z)
gdzie x oraz y są składnikami dodawania, natomiast z symbolizuje wynik tej operacji. Jeśli nastąpi wywołanie tej procedury za pomocą
Dodaj(2,3,z)
to z utożsami się z 5.
Natomiast po wywołaniu:
Dodaj(z,3,5)
rezultat wykonania tej procedury będzie równoważny odejmowaniu, w następstwie czego x utożsami się z 2.

* * *

Programowanie logiczne można — w pewnym sensie — traktować jako połączenie cech programowania funkcyjnego oraz relacyjnych baz danych. Wykorzystuje ono bowiem relacje do reprezentacji informacji oraz stosuje dedukcję w celu wyprowadzenia wniosków z założeń [12]. R. A. Kowalski uważa, że programowanie logiczne dostarcza brakującego ogniwa w ujednoliceniu języków programowania oraz języków zapytań systemów baz danych [10].

Wydaje się, że decydujące zmiany w praktyce programowania nastąpią z chwilą wprowadzenia komputerów tzw. piątej generacji. Częściowo na charakter tych zmian wskazują upowszechniające się systemy ze sztuczną inteligencją, zwłaszcza systemy ekspertowe. Może to spowodować przełamanie istniejącego obecnie paradygmatu programowania (opartego na algorytmicznych językach programowania) na rzecz programowania logicznego.

LITERATURA

[1] Amarel S.: On Machine Representations of Problems of Reasoning about Actions — The Missionaries and Cannibals Problem. Machine Intelligence, Vol. 3, pp. 131—171, B. Meltzer and D. Michie, Eds. Edinburgh University Press, 1966
[2] Emden M. H. van, Kowalski R. A.: The Semantics of Predicate Logic as a Programming Language, Journal of the ACM, Vol. 23, pp. 733—744, October 1976
[3] Green C. C.: The Application of Theorem Proving to Problem Solving. pp. 219—240, Proc. 1st Intern. Joint Conf. on Artificial Intelligence, Washington (D.C.), 1969
[4] Hayes P. J.: Computation and Deduction. Proc. 2nd Symposium on Mathematical Foundations of Computer Science, pp. 105—118, Czechoslovak Academy of Sciences, 1973
[5] Hogger C. J.: Introduction to Logic Programming, Academic Press, London, 1984
[6] Horn A.: On Sentences which are True of Direct Unions of Algebras, Journal of Symbolic Logic, Vol. 16, pp. 14—21, 1951
[7] Kluźniak F., Szpakowicz S.: Prolog, WNT, Warszawa, 1983
[8] Kowalski R. A.: Predicate Logic as a Programming Language. pp. 569—574, Proc. IFIP Congress 1974, North-Holland, Amsterdam, 1974
[9] Kowalski R. A.: Algorithm = Logic + Control. Conem. of the ACM, Vol. 22, pp. 424—431, 1979
[10] Kowalski R. A.: Logic Programming in the Fifth Generation. Proc. SPL Conference on The Fifth Generation Dawn of the Computer Age, London, July 1982
[11] Kowalski R. A.: Logic for Problem Solving. Elsevier, New York, 1985
[12] Kowalski R. A.: Directions for Logic Programming, Dept. of Computing, Imperial College, London, May 1985
[13] Robinson J. A.: Machine Oriented Logic Based on the Resolution Principle, Journal of the ACM, Vol. 12, pp. 23—41, January 1965.

Ceny ogłoszeń

Od 1 stycznia br. obowiązują następujące ceny ogłoszeń publikowanych na naszych łamach:

ogłoszenia duże (zależnie od objętości):
cała strona — 35 tys. zł, 3/4 — 30 tys., 1/2 — 25 tys., 1/4 — 20 tys., 1/8 — 15 tys.

ogłoszenia drobne (zależnie od liczby słów)
jedno słowo — 30 zł

Dodatki do ceny podstawowej:

— za dodatkowy kolor (na okładce) +30%
— za zamieszczenie ogłoszenia na czwartej stronie okładki +100%
— za zamieszczenie ogłoszenia na trzeciej stronie okładki +50%

Zniżki:

— za ogłoszenie 3—5-krotne —5%
— za ogłoszenia 6—10-krotne —10%
— za ogłoszenia 11-krotne i powyżej —20%
— za artykuły reklamowe i wkładki wykonane przez zleceniodawcę —40%
— za bloki i biuletyny wykonane przez zleceniodawcę — maks. —60%

W przypadku dostarczenia przez zleceniodawcę materiału ilustracyjnego nie odpowiadającego warunkom technicznym druku lub tekstu wymagającego redakcyjnego opracowania, do powyższych cen doliczane będą koszty odpowiednich usług fotograficznych, graficznych lub przygotowania tekstów.

Ogłoszenia • Ogłoszenia • Ogłoszenia • Ogłoszenia

BIURO USŁUG KOMPUTEROWYCH. Pośrednictwo sprzedaży mikrokomputerów, części zamiennych. Warszawa, tel. 41-44-48.

EO/272/K/86

Ogłoszenia • Ogłoszenia • Ogłoszenia • Ogłoszenia

PIOTR COFTA
Instytut Informatyki
Politechnika Gdańska

Norma IEEE — języki asemblerowe dla mikroprocesorów

Rozwój języków programowania przebiega niejednokrotnie w sposób mało skoordynowany. Szczególnie w wypadku języków asemblerowych wyraźnie widać wpływ przypadkowych koncepcji, walkę poszczególnych firm i inne niekorzystne zjawiska. Dlatego języki te charakteryzują się największą różnorodnością nie tylko pod względem dostępnych instrukcji (co jest zrozumiałe), ale także pod względem postaci zewnętrznej programu. Ponieważ sytuacja taka jest z wielu względów niekorzystna, amerykańska organizacja IEEE (Institute of Electrical and Electronics Engineers) podjęła próbę ujęcia formatu asemblerów w pewien jednolity standard. Prace te, prowadzone przez grupę roboczą P694, były częścią szerszego planu, z którego zrealizowano m.in. normę opisującą format kodu wynikowego [4] i normę opisującą operacje zmiennoprzecinkowe [3].

Prace nad normą obejmującą języki asemblerowe trwały kilka lat. Jako datę sformułowania początkowych założeń można przyjąć 1979 rok [2]. W 1985 roku zakończono pewną fazę prac, a ich efektem stała się norma **IEEE 694-1985**, opisana poniżej [1].

Celem normy jest wyeliminowanie niepotrzebnych różnorodności i niejednoznaczności występujących w sposobie zapisu postaci źródłowej programów w języku asemblera. Autorzy zdają sobie sprawę z tego, iż próby narzucenia jakiejkolwiek normalizacji sposobu pracy samego mikroprocesora są z góry skazane na niepowodzenie. Ponieważ jednak w językach asemblera obie te rzeczy pozostają w ścisłym związku, przyjęto rozwiązanie kompromisowe. Norma precyzuje zatem tylko pewne reguły postępowania, pomocne przy definiowaniu postaci źródłowej asemblera dla konkretnego mikroprocesora. W części, w której architektury współczesnych uniwersalnych mikroprocesorów są do siebie zbliżone, reguły te mają postać dosyć ścisłych zaleceń lub wykazów. W pozostałych wypadkach wskazówki zawarte w normie pozwalają dosyć łatwo interpretować zaistniałe sytuacje, zgodnie z jej intencjami.

Dążeniem autorów było, aby istnienie normy służyło eliminacji niejednoznaczności, skróceniu czasu potrzebnego na naukę języka, zmniejszeniu liczby popełnianych błędów oraz polepszeniu przenośności oprogramowania. Zgodnie z założeniami norma nie precyzuje stylu programowania, architektury mikroprocesora ani też sposobu implementacji translatora. Norma nie obejmuje również postaci instrukcji sterujących translacją warunkową lub też makroprzetwarzaniem. Tematyka ta stanowi przedmiot dalszych prac grupy roboczej.

POSTAĆ TEKSTU ŹRÓDŁOWEGO

Zgodnie z założeniami normy, poszczególne instrukcje w języku asemblera zapisywane są w oddzielnych liniach tekstu. Każda linia może zawierać (w podanej kolejności): etykietę, mnemonikę opisującą kod rozkazu, argumenty rozdzielone przecinkami oraz komentarz. Dozwolone są linie puste, linie zawierające tylko etykietę i (lub) komentarz. Poszczególne składowe linie muszą być oddzielone od siebie co najmniej jednym widocznym znakiem odstępu, np. znaki spacji, tabulacji itp.

Norma nie precyzuje dozwolonej maksymalnej długości etykiet, zezwalając jedynie na kończenie jej znakiem dwukropka (choć nie jest to wymagane). Etykieta składa się z liter i cyfr, przy czym pierwszy znak jest literą. Zezwolenie na użycie małych liter, a także decyzję co do rozróżniania małych i dużych liter pozostawiono do dyspozycji implementatora. Podobnie zaliczenie znaku podkreślenia do zbioru liter zależy od konkretnej implementacji.

Konieczne jest poprzedzenie komentarza znakiem średnika. Komentarz taki zajmuje miejsce aż do końca linii tekstu. Norma nie narzuca żadnych innych ograniczeń na używanie tej składowej języka.

WYRAŻENIA ARYTMETYCZNE CZASU KOMPILACJI

Jednym z istotniejszych elementów języka asemblera jest format wyrażeń arytmetycznych czasu kompilacji, używanych w zapisie argumentów instrukcji. Zalecenia normy są dosyć szczegółowe i obejmują:

a. Zestaw operacji. Jako minimalny określono zestaw zawierający operację dodawania oznaczoną znakiem "+" oraz operację odejmowania "—". Ponadto zalecono zaimplementowanie operacji: mnożenia, dzielenia całkowitoliczbowego, iloczynu logicznego, sumy logicznej oraz sumy modulo dwa. Zezwolono ponadto na implementację dowolnych innych operacji, przy czym ich zapis ma mieć postać zbliżoną do stosowanej w normie [4].

b. Użycie nawiasów. Wyrażenia mogą zawierać nawiasy okrągłe "(" i ")" służące do określenia kolejności wykonywania operacji.

c. Postać stałych liczbowych. Dozwolone jest użycie stałych czterech typów, przy czym każda z nich jest poprzedzona skrótem określającym podstawę liczby, np. dwójkowa — **B'10110**, dziesiętne — **D'12**, szesnastkowa — **H'A2**. Stałe liczbowe nie poprzedzone żadnym ze skrótów są interpretowane jako stałe zapisane w zastępczym systemie liczbowym.

d. Postać ciągu znaków (napisu). Napis jest ujęty w cudzysłowy "...", a sam cudzysłów, o ile ma być interpretowany jako znak napisu ma postać " ". Jeżeli nie jest jednoznaczny kod, według którego przyporządkowuje się znakom wartości liczbowe, to precyzuje go litera podana przed cudzysłowem, np. **A** oznacza kod **ASCII**. Zezwolono na podanie, bezpośrednio po napisie lub między napisami, liczby opisującej wprost kod jednego znaku. Liczba taka nie jest oddzielana żadnym znakiem od cudzysłowów ograniczających napis.

e. Odwołanie do wskaźnika lokacji. Przyjęto, iż w trakcie interpretacji całej instrukcji wskaźnik lokacji wskazuje zawsze adres pierwszego bajtu instrukcji. Odwołanie do wskaźnika następuje przez użycie gwiazdki.

f. W wyrażeniach zezwolono na użycie symboli (etykiet) wszędzie tam, gdzie mogły być użyte stałe liczbowe, chyba że konkretna implementacja ogranicza te możliwości.

MNEMONIKA INSTRUKCJI

Norma zawiera opis proponowanej **mnemoniki dla instrukcji** najczęściej wykonywanych przez współczesne mikroprocesory, np. **ADD, SUB, INC** itp. Jednocześnie proponowane jest systematyczne podejście do metodologii tworzenia takich nazw, pożyteczne w wypadku jakiejkolwiek nietypowości w liście rozkazów konkretnego mikro-

procesora. **Nazwa instrukcji (mnemonika) powinna być tworzona według następujących zasad ogólnych:**

a. Pierwszy znak nazwy musi być pierwszą literą angielskiego czasownika określającego operację wykonywaną przez instrukcję.

b. Tryb adresowania poszczególnych argumentów, a także ich położenie w strukturze mikroprocesora (np. identyfikator rejestru) nie może być określane przez mnemonikę.

c. Określenie warunku wykonania instrukcji (np. stan wskaźników w wypadku skoku warunkowego) powinno być widoczne w mnemonice. Dla instrukcji tego rodzaju zaleca się użycie nazwy odpowiadającej bezwarunkowemu wykonaniu instrukcji, uzupełnionej przyrostkiem określającym warunek. Zalecane przyrostki (jest ich 24) obejmują typowe sytuacje związane z testowaniem stanu wskaźników, np.: **Z** — gdy wynik jest równy zeru, **GT** — gdy jest „większy od" dla liczb ze znakiem, **L** — gdy jest „mniejszy" dla liczb bez znaku, **C** — gdy zawartość wskaźnika przeniesienia wynosi 1. Ponadto instrukcje tego rodzaju (szczególnie instrukcje skoku warunkowego) mogą posiadać przedrostek **D** lub **I** wskazujący, iż w trakcie wykonywania instrukcji następuje zmniejszenie lub zwiększenie zawartości pewnego argumentu (być może domyślnego).

d. Jeżeli istnieją odrębne instrukcje do wykonywania operacji na liczbach różnych typów (np. oddzielna instrukcja dzielenia dla liczb bez znaku i liczb ze znakiem), dozwolone jest użycie przyrostka rozróżniającego takie instrukcje. Podobnie przez odpowiedni przyrostek można wskazać, iż instrukcja wykonuje się w odwrotnej kolejności niż to przyjęto standardowo. Przy użyciu odpowiednich przyrostków można wskazać również, iż instrukcja w specyficzny sposób wykorzystuje wskaźniki, korzysta z całej grupy argumentów itp.

e. Typ argumentu (jego długość, stosowany sposób zapisu) może być umieszczony w nazwie, choć — jeżeli nie powoduje to niejednoznaczności — może być opuszczony. Jeżeli instrukcja ma kilka argumentów różnych typów, to typ wyniku może być umieszczony w nazwie, a typy poszczególnych danych — przy opisie argumentu. Dla określenia typu można użyć przyrostka poprzedzonego znakiem kropki. Tymczasowo zdefiniowany zestaw przyrostków obejmuje:

I lub **I1** — pojedynczy bit
In — n sąsiednich bitów
B — bajt (osiem bitów)
S — dwa bajty
L — cztery bajty
Q — osiem bajtów
D — liczba dziesiętna
P lub **DP** — liczba dziesiętna upakowana (o ile występują dwa formaty liczb dziesiętnych)
Fx — liczba zmiennoprzecinkowa, o postaci zgodnej z normą [3].

Interesująca jest możliwość użycia aż trzech rodzajów mnemoniki dla opisanego prostego przesyłania danych. **Mnemonika LD** zgodnie z normą **może opisywać przesyłanie dowolnego typu,** tj. z pamięci do rejestru, z rejestru do rejestru, między rejestrami i komórkami pamięci. Jednakże zalecane jest użycie jej tylko do opisania przesłania z pamięci do rejestru. Dla przesłania z rejestru do pamięci zalecana jest **mnemonika ST,** a dla pozostałych przesłań — **MOVE.**

Ponadto norma dopuszcza sytuację, w której kilka rodzajów mnemoniki opisuje tę samą instrukcję, co jest istotnym ułatwieniem, na przykład w wypadku skoków względnych.

ARGUMENTY

Norma nie precyzuje liczby ani typu argumentów związanych z konkretną instrukcją, uzależniając to od wymogów implementacji. Jednakże w normie sprecyzowano pewne zalecenia co do kolejności argumentów. **Zalecenia te precyzują, iż argumenty** opisujące miejsce przesłania wyniku lub wyników operacji **powinny być umieszczone przed argumentami opisującymi dane dla instrukcji. Wyjątek stanowi** jedynie **instrukcja ST.** Podobnie sprecyzowano, iż w wypadku instrukcji przesunięcia, dla których liczba bajtów nie jest wartością domyślną, liczba ta powinna być podana jako ostatni z argumentów. Zgodnie z ogólnie przyjętą w normie zasadą nienarzucania zbędnych wymagań, możliwe jest używanie argumentów domyślnych, nie opisywanych w instrukcji, o ile nie spowoduje to niejednoznaczności. Przykładem takiej sytuacji jest instrukcja umieszczania na stosie, w której niejawnym argumentem jest wskaźnik stosu, czy też instrukcje arytmetyczne, pozostawiające wynik zawsze w tym samym rejestrze.

Sam argument zapisywany jest jako wyrażenie arytmetyczne czasu translacji, uzupełnione zapisami wskazującymi na stosowany tryb adresowania oraz typ argumentu.

TRYB ADRESOWANIA

Jak wspomniano, mnemonika nie może zawierać opisu trybu adresowania, dlatego też umieszcza się go przy opisie poszczególnych argumentów. Jeżeli nie powoduje to niejednoznaczności, można w konkretnej implementacji zrezygnować z konieczności wskazywania niektórych trybów adresowania. Dlatego poniższe uwagi mają jedynie postać zlecenia.

Opis trybu adresowania składa się z przedrostka określającego sposób wyznaczania wartości argumentu oraz z przyrostków określających sposób wyznaczania adresu oraz określających typ argumentu. Przedrostek może przyjąć jedną z postaci:

— **określa,** iż wartością argumentu jest wartość wyrażenia arytmetycznego czasu kompilacji, a wartość ta może być umieszczona w dowolnym miejscu, np. wewnątrz instrukcji,

/ — **określa,** iż wartością argumentu jest zawartość pamięci pod adresem, którego wartość jest wskazana przez wyrażenie arytmetyczne,

. — **określa,** iż wartością argumentu jest zawartość rejestru o numerze lub symbolu określonym przez wyrażenie arytmetyczne,

$ — **określa,** iż wartość wyrażenia jest liczona względem aktualnej wartości wskaźnika lokacji (licznika rozkazów), a nie względem ustalonego początku pamięci,

! — **określa,** iż argument znajduje się w specyficznym obszarze pamięci (np. na stronie zerowej).

Przyrostek umieszczony po wyrażeniu arytmetycznym i ujęty w nawiasy kwadratowe "[" i "]" określa sposób indeksowania adresu. Wewnątrz nawiasów można podać wartości stałe oraz opisy rejestrów, biorących udział w wyznaczaniu adresu efektywnego. Zapisy umieszczone w nawiasach kwadratowych można zagnieżdżać. Całe wyrażenie arytmetyczne można również ująć w nawiasy kwadratowe, wskazując w ten sposób na użycie adresowania pośredniego.

Jeżeli w trakcie wykonywania programu, przy wyznaczaniu adresu efektywnego jeden z elementów adresu (np. rejestr) automatycznie ulega zmianie (tzw. autoinkrementacja i autodekrementacja), można to zasygnalizować, umieszczając obok niego znaki ++ lub −− oraz znaki określające wielkość tych zmian.

STEROWANIE TRANSLACJĄ

Norma poświęca stosunkowo niewiele miejsca sterowaniu procesem translacji, wyraźnie przedstawiając ten problem do rozwiązania implementatorom. Zdefiniowano w niej jedynie zestaw dyrektyw uznany za minimalny, odstępując jednakże od zasady sugerowania dalszych kierunków rozwoju mechanizmów sterowania. Minimalny zestaw dyrektyw sterujących translacją obejmuje:

ASSIGN — przypisanie wartości symbolowi z możliwością zmiany wartości; przypisywany jest także atrybut związany z wartością, np. rejestr

EQU — przypisywanie wartości bez możliwości zmiany

DATA — deklaracja danych wszelkich typów; typy danych są precyzowane przy konkretnych argumentach

RES — rezerwacja miejsca w pamięci bez nadania wartości początkowej

ORG — nadanie wartości wskaźnikowi lokacji

EXPORT — opis symboli udostępnianych innym segmentom

IMPORT — opis symboli pobieranych z innych segmentów
END — koniec segmentu lub programu
PAGE — wysuw papieru na wydruku do nowej strony
HEADING — specyfikacja tekstu nagłówkowego, drukowanego na każdej stronie
RADIX — ustalenie zastępczej podstawy dla stałych liczbowych bez zdeklarowanej podstawy.

PRZYKŁAD

Poniżej przedstawiono fragment programu przeznaczonego do wykonania na mikroprocesorze Intel 8080, przy czym w lewej kolumnie zaprezentowano go w postaci zgodnej ze standardem proponowanym przez firmę Intel, a w prawej — zgodnie z normą IEEE. Tekst przykładu zaczerpnięto z [5], nieznacznie go modyfikując. Usunięcie komentarzy, ze względu na identyczny sposób ich zapisu, nie wpływa na wartość przykładu, zwiększając jego czytelność.

```
        LXI   H,ADDRESS        LD   .HL,#ADDRESS
        MVI   C,RF+IRQ         LD   .C,#RF+IRQ
        MVI   B,CB+ND+IRQ      LD   .B,#CB+ND+IRQ

READ:   IN SB EG               READ: IN  .A,SREG
        CMP   B                      CMP .A,.B
        JNZ   RESULT                 BNE /RESULT

DATA:   IN    DREG             DATA: IN  .A,DREG
        MOV   M,A                    ST  .A,[.HL]
        INX   H                      INC .HL
        JMP   READ                   BR  /READ

RESULT: CMP C                  RESULT: CMP .A,.C
        JNZ READ                       BNZ /READ
```

* * *

Zgodnie z założeniami, norma jest przeznaczona do wspomagania projektowania asemblerów dla mikroprocesorów. Jednakże reguły w niej zaprezentowane mogą być pomocne również dla programistów dużych komputerów, choć należy liczyć się z pewnymi kłopotami, związanymi głównie z odmiennością architektury, a więc — formatów danych.

Jak łatwo można dostrzec, norma precyzuje tylko niewielki zestaw środków udostępnianych programiście przez współczesne asemblery. Brak w niej m.in. opisu mechanizmu makroinstrukcji, kompilacji warunkowej itd. W tym też kierunku idą obecne prace grupy roboczej IEEE. Kolejnym ich celem jest opracowanie normy języka makroasemblera, obejmującej specyfikację nie opracowanych dotychczas środków programowych. Prace znajdują się obecnie (styczeń 1986) na etapie precyzowania podstawowych założeń nowej normy.

Oczywistym kierunkiem prac podejmowanych w związku z normą jest także opracowywanie jej implementacji dla konkretnych mikroprocesorów, co prowadzi do precyzowania przedstawionych zaleceń. Jeden z pierwszych przedstawionych projektów dotyczy postaci „znormalizowanego" języka asemblera dla mikroprocesora Z80. Wynika to niewątpliwie nie tylko z ciągłej popularności tego procesora, lecz także z faktu istnienia dla niego aż trzech asemblerów różniących się formatem postaci źródłowej. W przygotowaniu znajduje się także podobna specyfikacja dotycząca procesorów 16-bitowych firmy Intel.

Kolejny kierunek prac, podejmowanych w związku z pojawieniem się normy, obejmuje próby budowy translatorów akceptujących konkretne języki zgodnie z normą, a także próby budowy narzędzi umożliwiających łatwe konstruowanie takich translatorów. Jedną z pierwszych prac w tej dziedzinie jest opisana w artykule [6], obecnie zaś trwająca praca nad translatorem dla Z80.

Działania podejmowane w związku z pojawieniem się normy zataczają coraz szersze kręgi. W chwili obecnej można już mówić o akceptacji propozycji ujętych w normie, a dalszy rozwój prac niewątpliwie umocni jej pozycję.

LITERATURA

[1] Baldwin G.: Towards an Assembly Language Standard. IEEE Micro, Vol. 4, No. 4, pp. 81—85, August 1984
[2] Fischer W.: Microprocessor Assembly Language Draft Standard. IEEE Computer, Vol. 12, No. 12, pp. 96—109, December 1979
[3] IEEE: Binary Floating-Point Arithmetic. IEEE Std. 754-1985, Institute of Electric and Electronics Engineers, New York (NY), 1985 (por. Informatyka, nr 1, str. 12, 1985)
[4] IEEE: Microprocessor Uniwersal Format for Object Modules. IEEE Std. 695-1985, Institute of Electrical and Electronics Engineers, New York (NY), 1985
[5] Pluta A.: Współpraca z napędem dysków elastycznych. INFORMATYKA, nr 2, str. 15, 1985
[6] Skordalakis E.: Towards a Quick Implementation of Assemblers for the Proposed IEEE Microprocessor Assembly Language Draft Standard. Microprocessing and Microprogramming, Vol. 12, No. 2, p. 89—97, September 1983.

PRZEDSIĘBIORSTWO ZAGRANICZNE

DIALOG

oferuje:

- komputery osobiste (PC) klasy profesjonalnej
- moduły dodatkowe do komputerów osobistych
- oprogramowanie

Zapewniamy krótkie terminy dostaw oraz konkurencyjne ceny.
Szczegółowe informacje handlowe, dane techniczne oraz ceny przesyłamy pocztą.

PZ DIALOG
96-313 Jaktorów
Chylice 5
woj. skierniewickie
tel. Warszawa 55-24-24

EO/599/K/86

JERZY KARCZMARCZUK
Instytut Fizyki
Uniwersytet Jagielloński
Kraków

Język programowania ICON (3)

W ostatniej części artykułu dokończono omawianie generatorów oraz przedstawiono operacje tekstowe w Iconie.

Generatory i zasada celowości

Każde wyrażenie zawierające w sobie generator samo może generować niebanalną sekwencję wynikową. Właściwości generowania propaguje się tak samo jak niepowodzenie. Przykładowo, wyrażenie:

(1|2)+(3|5)

generuje sekwencję 4, 6, 5 i 7. Z kolei wyrażenie relacyjne takie jak **upto(s1, s2) > 10**
generuje podzbiór sekwencji wynikowej samego **upto;** jej część zostanie odrzucona wskutek niepowodzenia operacji ">".

Charakterystyczną cechą Iconu jest „zasada celowości", polegająca na tym, że mechanizm ewaluujący wyrażenia zawsze dąży do otrzymania jakiegoś wyniku, jeśli jest to w ogóle możliwe. Wyrażenie:

write(10<upto(s1,s2))

spowoduje wydrukowanie pierwszej pozycji dostarczonej przez funkcję **upto, większej od 10.** Poprzednie pozycje będą generowane, ale odrzucone, więc na mocy „zasady celowości" działanie generatora będzie automatycznie wznawiane aż do skutku lub ostatecznego niepowodzenia. Jeśli wyrażenie zawiera dwa generatory:

... gen1 ... gen2

i cała sekwencja **gen2** zostanie wyczerpana bez ostatecznego dostarczenia wyniku, to nastąpi nawrót do **gen1,** wygenerowanie następnego elementu jego sekwencji i powtórne uruchomienie **gen2** od początku (w kontekście być może już zmienionym przez gen1). Nie jest do tego potrzebny żaden jawny iterator. Szczególnym przykładem tej strategii ewaluacyjnej jest postępowanie podczas wywołania procedury, **np. fun(E1,E2).** Jeśli ewaluacja **E2** zakończy się niepowodzeniem, a **E1** jest generatorem, to jego działanie jest wznawiane, a następnie ewaluacja **E2** jest powtarzana. Nawroty są ograniczone do pojedynczych wyrażeń, np. niepowodzenie **E2** w konstrukcji **E1; E2** nie spowoduje nawrotu do **E1.**

Ta zasada, określona po angielsku jako goas-oriented programming, jest jednym z najważniejszych mechanizmów sterujących w Iconie. W Snobolu jest ona ograniczona do operacji porównania napisu z wzorcem, stanowiąc niejako „język w języku". W Iconie jest integralnym elementem ogólnej strategii ewaluacyjnej. Warto dodać, że zasada goal-oriented programming stanowi również jedną z podstaw programowania w Prologu.

Procedury jako generatory

Konstruowanie własnych złożonych generatorów jest w Iconie łatwe. Wystarczy napisać procedurę, która zamiast słowa **return** będzie zawierała specjalne słowo **suspend.** Konstrukcja:

suspend E

spowoduje powrót sterowania do miejsca wywołania z wartością **E** jako wynikiem, jednak procedura nie zakończy całkowicie swojej pracy; pozostanie niejako w zawieszeniu i będzie można wznowić jej działanie przez nawrót, sterowany iteratorem lub wymuszony niepowodzeniem ewaluacji wyrażenia zawierającego wywołanie: Wynika stąd, że generator będzie znacznie bardziej obciążał pamięć niż zwykła, nawet rekursywna procedura. Poniżej podano przykładową procedurę, która jako swoją sekwencję wynikową dostarcza kolejnych liczb ciągu Fibonacciego:

```
procedure Fibgen ( )
  local i,j,k
  repeat {suspend k := i + j
i := j; j := k}
end
```

Wyrażenie:

every write (Fibgen ( ) 1/20) [1]

spowoduje wydrukowanie pierwszych 20 elementów ciągu. Należy zauważyć, że nawrót do zawieszonego generatora jest tylko powtórnym przekazaniem sterowania (ang. control backtracking), natomiast takie operacje jak przypisanie wartości są nieodwracalne. Umożliwia to użycie lokalnych zmiennych **i** oraz **j** w procedurze Fibgen do przechowania wartości między kolejnymi wznowieniami. Zwolennicy Prologu mogą stwierdzić, że tak wspaniałe narzędzie, jakim jest możliwość wycofania się programu z pewnej drogi decyzyjnej, jest w Iconie zupełnie stępione, gdyż właśnie możliwość nawrotów z odtworzeniem pełnego kontekstu ewaluacji klauzuli Prologu umożliwia ich statyczną interpretację i stanowi o sile języka (i o jego popularności, mimo „zakucia go" w nieprzyjemny pancerz klauzul Horna). Jednak Icon zawiera również operator przypisania odwracalnego, oznaczonego jako <—, a także odwracalną zmianę: <—>. Wyrażenie:

(i <—j) > k

spowoduje przypisanie wartości **j** zmiennej i tylko wtedy, gdy jest ona większa od **k.** W przeciwnym wypadku niepowodzenie spowoduje wycofanie przypisania. Tak więc, na życzenie można programować w stylu bardziej zbliżonym do Prologu. W następnym punkcie omówiono inny dostępny w Iconie sposób automatycznego odtwarzania kontekstu po nawrocie (ang. data backtracking), który stanowi element tzw. operacji skanowania tekstu.

Przykładem zastosowania odwracalnego przypisania jest procedura rozwiązująca klasyczny problem ośmiu hetmanów. Chodzi w nim o umieszczenie na szachownicy ośmiu nie szachujących się wzajemnie hetmanów. Klasyczna metoda postępowania polega na kolejnym stawianiu hetmanów w wolne miejsca, a jeśli następny, np. szósty, hetman już nie ma gdzie stanąć, wycofuje się poprzedni, tj. piąty,

[1] Drukarnia nie ma w składzie znaku *backslash* (lewa kreska ukośna), który w tym artykule zastępujemy literą l/.

Dr JERZY KARCZMARCZUK ukończył studia fizyczne na Uniwersytecie Jagiellońskim w 1970 roku. Pracuje jako adiunkt w Zakładzie Teorii Cząstek Instytutu Fizyki i zajmuje się głównie teoretyczną fizyką wysokich energii. Ma spore doświadczenie dydaktyczne. Interesuje się implementacją języków programowania, mikrokomputerami — zwłaszcza ich oprogramowaniem podstawowym i zastosowaniami w dydaktyce, a także komputerowymi rachunkami algebraicznymi.

stawia go gdzie indziej i kontynuuje ustawianie. Jeśli żadne ustawienie piątego hetmana nie jest skuteczne, to wycofuje się również czwartego itd. Zainteresowani czytelnicy mogą porównać poniższą konstrukcję z algorytmem w Pascalu opisanym np. w książce [2] (str. 155).

W poniższym rozwiązaniu rzędy szachownicy są reprezentowane przez listę 8-elementową, inicjalizowaną wyrażeniem:

row := list(8,0).

Zero oznacza, że odpowiedni rząd jest pusty. Przekątne są podzielone na dwie grupy — od lewej w górę oraz od lewej w dół. Każda grupa zawiera 15 przekątnych i jest reprezentowana przez listę 15-clementową, nazywaną odpowiednio **up** i **down**. W liście **up**, element **nr 1** odpowiada przekątnej składającej się z pola **a8**, **nr 2** przechodzi przez pola **a7** i **b8** itd. Podobnie w liście **down** pierwszy element odpowiada przekątnej, przechodzącej przez pole **h8**, drugi — przez **g8** i **h7** itd., aż do **a1**. Hetmany są numerowane kolumnami, w których będą umieszczane — numery od 1 do 8 odpowiadają kolumnom od **a** do **h**. Hetman może być umieszczany w kolumnie **c** i rzędzie **r** jeśli zachodzi następująca relacja:

row [r] = down [r + c — 1] = up [8 + r — c] = 0

Po umieszczeniu hetmana w danym polu, odpowiednie elementy list **row, up** i **down** przyjmą wartości różne od zera. Wpisywaną wartością jest dla wygody numer wiersza. Poniżej przedstawiono cały program, razem z drukowaniem wygenerowanych wyników.

```
procedure main ()
  write (q(1),q(2),q(3),q(4),q(5),q(6),q(7),q(8))
end
procedure q(c)
  suspend place (1 to 8, c) # pierwszy parametr jest rzędem
end
procedure place (r,c)
  static up, down, row
  initial {up := list(15,0); down := list(15,0)
  row := list(8,0)}
  if row [r] = down [r + c — 1] = up [8 + r — c] = 0
  then suspend row [r] <— down [r + c — 1] <— up [8 +
+ r — c] <— r
end
```

Wydrukowane zostanie jako rozwiązanie: 15863724. Oczywiście, umieszczając instrukcję **write** pod iteratorem **every** otrzymamy wszystkie 92 rozwiązania (12 powtórzone 8 razy z odbiciami i obrotami). Odwracalne przypisania spowodują, że w wypadku konieczności wycofania hetmana, automatycznie zniknie informacja o jego poprzednim ustawieniu.

Ko-wyrażenia

Ko-wyrażenia (ang. co-expressions) stanowią jeszcze jeden niezwykle mocny i oryginalny mechanizm sterowania w Iconie. Pojęcie to jest potomkiem tzw. wyrażenia odroczonego w Snobolu, ale w kontekście związanym z generatorami nabiera nowych barw. Iteracja oraz nawroty po niepowodzeniu nie zawsze są wystarczające lub odpowiednio wygodne, aby w sposób kontrolowany przejrzeć sekwencję wynikową generatora. Czasami pożyteczne jest dysponowanie mechanizmem dostarczającym kolejnego generowanego wyniku w innym, dowolnym kontekście, podobnie jak to ma miejsce w wypadku wywoływania procedur — mogą być użyte praktycznie gdziekolwiek w tekście programu. Tu jednak nie chodzi o uruchomienie generatora od początku, lecz o wznowienie jego pracy po zawieszeniu. Jest to możliwe dzięki oddzieleniu procesu tworzenia (kreacji) **ko-wyrażenia** od jego aktywacji.

Do kreacji służy konstrukcja:

create E

gdzie **E** jest wyrażeniem (na ogół generatorem). Przykładowo,

create find(s1,s2)

tworzy **ko-wyrażenie** związane z **find**. Można je przypisać jako wartość dowolnej zmiennej. Procedura **find** nie zostaje tutaj wywołana. Tworzone jest natomiast odniesienie (jest to coś bardziej złożonego niż zwykły odsyłacz, tj. adres) do kodu programu, który ewaluuje wywołanie:
find(s1,s2)

W wypadku **ko-wyrażenia** związanego z wywołaniem procedury użytkownika, tworzone są również kopie zmiennych lokalnych. Aktywacja **ko-wyrażenia** następuje wskutek wykonania operacji oznaczanej jednoargumentowym operatorem **u**. Pierwsza aktywacja jest równoważona ewaluacji argumentu **create**, podczas tworzenia tego **ko-wyrażenia**. Każda następna jest wznowieniem. Tak więc, jeśli wartością zmiennej **e** jest **ko-wyrażenie:**

create find(s1,s2),
to wyrażenie:
while write (u e)
jest równoważne
every write (find)s1,s2)).

Ko-wyrażenie można „odświeżyć", tj. spowodować, że sekwencja wynikowa związanego z nim generatora będzie tworzona od początku. Służy do tego operacja **^e**. Tworzy ona kopię **ko-wyrażenia e** ze wszystkimi atrybutami przywróconymi do stanu, który istniał już po kreacji **e**.

Przykładem użycia generatorów i **ko-wyrażeń** jest procedura generująca wszystkie permutacje ciągu listy **N** elementów. Użyty algorytm można opisać jednym zdaniem: wszystkie permutacje **N** obiektów tworzy się generując wszystkie permutacje początkowego podzbioru **N—1** elementów, następnie dołączając do każdej **N-ty** obiekt dla wszystkich możliwych pozycji w liście.

```
procedure wstaw(x,1) # wstaw obiekt x do listy 1
  if l === [ ] then return [x]
  suspend ([x] ||| l) | ([l [1] ||| wstaw(x,l [2 : 0])
end
procedure permutacja(l)
  local z
  if *l = 1 then return l
  z := pop(l)
  suspend wstaw(z,permutacja (l))
end
...
e := create permutacja ("a","b","c","d")
```

Tak więc każda aktywacja **u** e wygeneruje nową permutację. **Ko-wyrażenia** stanowią potężne narzędzie, przy użyciu którego można modelować najrozmaitsze struktury sterowania. Przykładowo, znany **iterator every E1** do **E2** można zrealizować za pomocą następującej procedury:

```
procedure EVERY (e1,e2) # e1, e2 są ko-wyrażeniami
  while ue1 do ue2
end
```

Z kolei, poniższa procedura realizuje naprzemienne generowanie wyników przez dwa generatory:

```
procedure inter (e1,e2)
  suspend | u(e1 | e2)
end
```

Sekwencją wynikową dla wywołania:

inter(create 1 to 3, create 6 to 10)
jest następująca: 1, 6, 2, 7, 3, 8, 9, 10.

Ko-wyrażenia mogą aktywować się nawzajem i przekazywać sobie parametry. Przez słowo kluczowe **& source** dostępne jest implicite odniesienia do **ko-wyrażenia**, które aktywowało bieżące **ko-wyrażenie**. Można w ten sposób zorganizować złożony system komunikujących się współprogramów (lub ko-procedur, stąd nazwa „ko-wyrażenia"). Ten element języka jest więc bardzo użyteczny w symulowaniu procesów współbieżnych.

PRZETWARZANIE TEKSTÓW

Przy analizie złożonych strukturalnie tekstów, funkcje omówione w części 2 mogą stanowić jądro dość skomplikowanej i wyrafinowanej strategii postępowania. Towarzyszy jej jednak sporo dość mechanicznej i niewdzięcznej pracy, gdyż trzeba umieścić tekst w argumencie odpowiedniej procedury przeszukującej, np. **many**, następnie wyciąć określone fragmenty, skleić z innymi, itp. Trzeba również zaprogramować odpowiednie pętle iteracyjne indeksu-

jące przeglądany tekst. Ten aspekt programowania można jednak w Iconie znacznie uprościć.

W Snobolu istnieje specjalna struktura składniowa zwana instrukcją porównania z wzorcem. Umożliwia ona pobranie interesującego ciągu znaków zwanego napisem podmiotowym i poddaniu go operacji „przykładania" wzorca, niekiedy bardzo złożonej. Wzorzec jest konstruowany jako struktura danych, ale ma sens operacyjny, tzn. steruje pewną procedurą systemową zwaną skanerem. Napis podmiotowy jest niejawnym argumentem procedury skanującej. Programista może skoncentrować uwagę na tym, co zamierza zrobić z napisem podmiotowym, a nie na jawnym programowaniu pętli indeksującej jego znaki.

W Iconie istnieje podobna operacja skanowania tekstu, ale rolę wzorca może pełnić dowolne wyrażenie. Operacja ta ma postać:

text ? E

gdzie **E** jest wyrażeniem zawierającym na ogół wywołania funkcji analizujących teksty. Jej lewostronnym argumentem jest tekst, ale po obu stronach mogą wystąpić generatory. Wynikiem tej operacji jest wynik ewaluacji wyrażenia **E** w kontekście dostarczanym przez tekst. Przykładowo, ewaluacja wyrażenia:

x := (text ? upto)'aeiouy'))

spowoduje dostarczenie numeru pozycji pierwszej samogłoski w tekście **text** lub zakończy się niepowodzeniem. Warto zauważyć, że funkcja **upto** jest tutaj wywołana z jednym argumentem. W operacji skanowania drugi argument **upto** jest przekazywany niejawnie. Możliwy jest również pośredni, ale jawny dostęp do tekstu podmiotowego za pośrednictwem słowa kluczowego **& subject**. Z kolei, słowo kluczowe **& pos** ma znaczenie pozycji kursora podczas skanowania tekstu. Można odczytać jego wartość lub ją zmienić, ale zazwyczaj służą do tego wbudowane funkcje analizujące tekst, co uwalnia użytkownika od konieczności jawnego konstruowania pętli przebiegającej znaki podmiotu.

Przykładami takich funkcji przesuwających kursor są generatory **tab** i **move**. Wywołanie funkcji **tab(i)** dostarcza początkowego fragmentu podmiotu: **& subject [& pos : i]** i przesuwa kursor do pozycji i. W wypadku niepowodzenia funkcja **tab** nie zmienia oczywiście pozycji kursora, jednak, co jest bardzo istotne, jeśli wywołanie **tab** jest fragmentem większego wyrażenia i wskutek niepowodzenia jest reaktywowane, odtwarza poprzednią wartość **& pos**. Działanie funkcji **move** jest podobne do **tab**, przy czym wartość argumentu dodaje się do **& pos**, a więc pełni rolę pozycji względnej i może być ujemna. Ujemny lub zerowy argument funkcji **tab** oznacza znany konwencjonalny sposób indeksowania od tyłu. Wywołanie **move(i)** dostarcza wartości:

& subject [& pos : & pos + i].

Często spotykaną konstrukcją jest wyrażenie:

tab(match(s))

które przystaje do fragmentu tekstu identycznego z s i przesuwa kursor poza ten fragment. Dla wygody wyrażenie to zapisuje się jako =s.

Rozważmy następujące zadanie: jeśli analizowany tekst zawiera więcej niż dwa znaki, to należy dostarczyć trzeci znak, w przeciwnym wypadku — cały podmiotowy tekst. Jego rozwiązaniem jest wyrażenie:

text ? ((move)2) & move(1)) | tab(0)

Jeśli tekst podmiotowy zawiera przynajmniej trzy znaki, to oba wywołania **move** powiodą się i drugie wywołanie dostarczy końcowego wyniku. Jeśli tekst zawiera tylko dwa znaki, to drugie wywołanie **move** kończy się niepowodzeniem, a pierwsze jest reaktywowane i aczkolwiek nie może już wygenerować nowego wyniku, odtwarza pierwotną wartość & pos, co powoduje, że wywołanie drugiego członu alternatywy dostarcza prawidłowego wyniku końcowego.

Argumenty operacji skanowania mogą być dowolnie skompilowane i mogą zawierać struktury sterowania. Ewaluacja wyrażenia:

tex ? while tab(upto('aeiouy')) do write(move(1))

powoduje wypisanie wszystkich małych samogłosek w tekście. Poniższy fragment programu wczytuje plik wejściowy złożony z wierszy, zawierających po dwa słowa (dowolne ciągi znaków) oddzielone dwukropkiem. Wiersz jest rozpakowywany, a każde dwa słowa są drukowane w dwóch wyrównanych, z lewej strony 10-kolumnowych szpaltach.

```
while line := read() do
  line ? write
(left(tab(upto(':')),10),
(move(1) & left(tab(0),10)
)
```

Ostatnim przykładem jest program tłumaczący wyrażenia arytmetyczne z tradycyjnej, infiksowej postaci na postać przedrostkową według standardowego schematu pierwszeństwa operatorów. Choć nie użyto tu żadnego wymyślnego, eleganckiego algorytmu, jednak zastosowana technika programowania jest dość niekonwencjonalna. Konwersji dokonuje rekursywna procedura **fix(E)**, która wyrażenie

E o postaci:
E1 op E2

przepisuje w formie:

op(fix(E1),fix(E2)).

Przykładowe wyrażenie:

x + 3 * y — u / n↑e↑2

zostanie przepisane w postaci:

+ (x,—(*(3,y), / (u,↑(n,↑(e,2)))))

Znak ↑ oznacza potęgowanie, dopuszczalne są również nawiasy.

Głównym problemem jest rozpoznanie i oddzielenie operatora wiodącego i jego argumentów. Gdy to już jest gotowe, ostatecznej konstrukcji wyniku dokonuje procedura form:

```
procedure form(arg1,op,arg2)
 return op || "(" || fix(arg1) || "," || fix(arg2) || ")"
end
```

Wiodącym operatorem jest operator o najniższym pierwszeństwie, tj. "+" albo "—". W wypadku występowania kilku takich samych operatorów, należy wybrać położony najbardziej na lewo, spośród operatorów łączących w prawo (np. potęgowanie) lub położony najbardziej na prawo operator łączący w lewo. Pierwsze zadanie jest łatwiejsze, jednak należy prawidłowo ominąć operatory zanurzone w podwyrażeniach opatrzonych nawiasami:

```
procedure rightop(exp,op)
  return ex ? form(tab(bal(op)),move(1),tab(0))
end
```

Wywołanie
tab(bal(op))

dostarcza początkowego fragmentu wyrażenia, tj. lewego argumentu operatora. W jednoargumentowym wywołaniu **bal** niejawnymi argumentami są: **& subject** oraz **para** "()". Lokalizacja operatora położonego najbardziej na prawo jest bardziej złożona. W tym celu generuje się całą sekwencję pozycji operatora i zatrzymuje ostatni wynik:

```
procedure leftop(exp,op)
  local j
  return exp ? {every j := bal(op)
form(tab(1/j),move(1),tab(0))}
end
```

Wyrażenie 1/j zawodzi, jeśli wartością j jest **& null**, tj. jeśli nie ma poszukiwanego operatora; w przeciwnym wypadku dostarcza wartości **j**. Kolejne wznowienie **bal** dostarcza coraz dłuższych lewych fragmentów **exp**, aż zostanie znaleziony ostatni operator.

Główna procedura usuwa ewentualne pary nawiasów obejmujące wyrażenia i wywołuje funkcje **leftop** i **rightop** w kolejności wzrastającego pierwszeństwa operatorów:

```
procedure fix(exp)
  while exp ?:= 2(="(",tab(bal(')')),pos(—1))
  return leftop(exp,'+—' | '*/') | rightop(exp,'↑') | exp
end
```

W pierwszym wierszu tej procedury usuwane są z tekstu **exp** wszystkie zewnętrzne pary nawiasów przez roszerzoną operację przypisania połączoną ze skanowaniem. Wyrażenie =")" powoduje przeskok przez lewy nawias, a **tab(bal(')'))** przez wszystkie inne znaki aż do zewnętrznego prawego nawiasu. Wywołanie funkcji wbudowanej **pos(i)** powoduje sprawdzenie, czy aktualna pozycja kursora wynosi i — w tym wypadku jest to pozycja na lewo od ostatniego znaku. Na uwagę zasługuje dowcipna konstrukcja: **2(E1,E2)**. Ogólnie, konstrukcja wyglądająca jak wywołanie procedury, lecz zawierająca liczbę całkowitą **n** zamiast nazwy procedury jest ewaluowana w ten sposób, że obliczane są wartości wszystkich wyrażeń w nawiasach z ewentualnymi nawrotami, a następnie jako wynik dostarczana jest **n-ta** wartość z listy. Pętla **while** służy do usunięcia wszystkich zewnętrznych par nawiasów. Drugi wiersz procedury powoduje przeszukanie wyrażenia i jego podwyrażeń na wszystkich poziomach. Najpierw poszukiwane są operatory addytywne, następnie multiplikatywne, potęgowanie, wreszcie wyrażenie jest zwracane niezmienione.

* * *

Autor zdaje sobie sprawę, że to opracowanie jest bardzo niekompletne. Ma ono na celu jedynie zinteresowanie czytelników tym językiem. Jeśli to się nie uda, należy winić autora, a nie Icon. Sam język jest znakomitym przykładem pewnej (oczywiście nie jedynej) tendencji rozwoju języków programowania o pascalopodobnym kształcie. Z jednej strony wykazuje, że w błędzie są ci, którzy uważają, iż główny postęp będzie polegał jedynie na wyposażeniu języka w coraz większą liczbę wbudowanych procedur i operatorów, z drugiej strony uwidacznia, że aby programować w stylu funkcjonalnym lub w stylu „logicznym" zbliżonym do rachunku predykatów, nie jest konieczne przyjęcie sztywnej i niezbyt przyjaznej składni Lispu lub Prologu.

Oczywiście Icon ma wiele wad, częściowo odziedziczonych po Snobolu. Brak statycznego sprawdzania typów zmiennych daje użytkownikowi olbrzymie możliwości, ale również okazję do popełniania bardzo trudnych do zlokalizowania błędów. Rozróżnienie między kontekstem, w którym występuje niepowodzenie, oraz błędem wykonania, nie jest zupełnie oczywiste.

Obecnie w Polsce znajdują się już taśmy dystrybucyjne Iconu i można się spodziewać, że niedługo zostanie on zainstalowany i oddany pod osąd szerszej publiczności. Wydawnictwo Naukowo-Techniczne przygotowało do druku tłumaczenie podręcznika Griswoldów. W czasopismach Software oraz ACM Transactions on Programming Languages and Systems można znaleźć szereg znakomitych artykułów o Iconie, napisanych przez jego twórców. Szczególnie godny odnotowania jest artykuł [1], gdzie opisano organizację pamięci oraz sposób przekazywania sterowania w języku umożliwiającym nawroty.

LITERATURA

[1] Griswold R. E., Wampler S. B.: The Implementation of Generators and God-Oriented Evaluation in Icon. Software, Vol. 13, pp. 495—518, 1983

[2] Writh N.: Algorytmy + struktury danych = programy. WNT Warszawa, 1980.

PAWEŁ GRZEGORZEWICZ
Instytut Informatyki
Politechnika Warszawska

Mechanizm półautomatycznego generatora poleceń dla systemu operacyjnego RSX-11M

Sterowanie współczesnym systemem operacyjnym wymaga stosowania złożonych języków poleceń. Polecenia te są zazwyczaj wprowadzane bezpośrednio z klawiatury. Metoda ta zapewnia maksymalną swobodę w wykorzystywaniu funkcji systemu oraz w kontrolowaniu na bieżąco procesu przetwarzania. Jednakże, stosowanie dużej liczby różnorodnych poleceń wraz z wieloma parametrami i opcjami utrudnia dokonanie prawidłowego wyboru oraz zwiększa długość wprowadzanych tekstów i prawdopodobieństwo wystąpienia błędów.

Mgr inż. PAWEŁ GRZEGORZEWICZ urodził się w 1961 roku w Warszawie. Studia wyższe ukończył w 1985 roku na Wydziale Elektroniki Politechniki Warszawskiej w Instytucie Informatyki. Obecnie pracuje w wymienionym Instytucie, specjalizując się w dziedzinie programowania współbieżnego oraz konstruowania systemów operacyjnych.

Skrócenie czasu przetwarzania można osiągnąć przez usprawnienie wprowadzania poleceń do systemu. Jest to osiągane przez funkcje makrogeneracji poleceń. Za pomocą tych funkcji można zbudować nowy poziom wirtualny systemu. Proste polecenie wydane na tym poziomie powoduje wygenerowanie i wykonywanie sekwencji poleceń z niższego poziomu, tj. z poziomu poleceń dla systemu operacyjnego. Na dodanym poziomie system cyfrowy może zachowywać się aktywnie, tzn. zadawać pytania osobie, która nim steruje.

W systemie RSX-11M dostępny jest prosty w użyciu i elastyczny mechanizm do półautomatycznego generowania poleceń. Są one generowane przez uprzywilejowany program na podstawie wzorca umieszczonego w pliku. Po wprowadzeniu z klawiatury

IDENT_PLIKU_Z_WZORCEM

program ten zaczyna interpretować zawartość pliku. W najprostszym przypadku plik może zawierać sekwencję poleceń dla systemu RSX. Oprócz prostego przesłania poleceń do wykonania, program rozpoznaje dodatkowe informacje sterujące interpretacją. W czasie interpretacji istnieje możliwość deklarowania zmiennych, etykiet oraz wykonywania instrukcji warunkowych, skoku, przypisania, wejścia-wyjścia i zmiany kontekstu interpretacji.

Język opisu wzorca poleceń

Język opisu wzorca umożliwia zorganizowanie konwersacji użytkownika z systemem RSX, a także prosty zapis modyfikowanych parametrami sekwencji poleceń. Po włączeniu opcji makrogeneracji instrukcją

.ENABLE SUBSTITUTION

w każdej interpretowanej linii, ciąg liter zawarty pomiędzy dwoma apostrofami jest traktowany jako nazwa zmiennej i zastępowany tekstowo wartością tej zmiennej. Zmienne są trzech rodzajów:

— tekstowe do 80 znaków
— logiczne
— numeryczne (całkowite nieujemne)..

Zestawienie właściwości typów zmiennych zawarto w tabeli 1. Zmienna i jej typ są definiowane przy pierwszym nadaniu wartości. Wartości są nadawane zmiennym przez instrukcje wejścia-wyjścia oraz przypisania. Instrukcja we-wy składa się z nazwy instrukcji (innej dla każdego typu), akceptowanego zakresu wartości, nazwy zmiennej podstawianej oraz tekstu zachęty, który pojawia się na terminalu w czasie wykonywania instrukcji. Przy próbie wprowadzenia wartości spoza akceptowanego zakresu — sygnalizowany jest błąd, a instrukcja jest wykonywana ponownie. Dozwolone postacie instrukcji we-wy oraz przypisania zestawiono w tabeli 2.

Ze względu na stosowanie deklaracji etykiet oraz instrukcji skoku, wzorce do generowania poleceń dla RSX przypominają programy zapisane w językach niestrukturalnych. Instrukcje i inne informacje sterujące dla interpretera rozpoczynają się od znaku kropki lub znaków specjalnych. Napisy nie rozpoczynające się od znaków specjalnych są traktowane jako polecenia dla RSX, tzn. są bezpośrednio przekazywane do wykonania. W tabeli 3 przedstawiono zestaw instrukcji rozpoznawanych przez interpreter wzorca do generowania poleceń.

W czasie interpretowania pliku poleceń możliwe jest wykorzystanie pewnych parametrów interpretacji. Składnia

Tabela 1. Zestawienie typów zmiennych i ich elementarnych właściwości

Właściwości typu	Typ logiczny	Typ liczbowy	Typ tekstowy
Zakres wartości	TRUE FALSE	ø..65536	ø..8ø znaków alfanumerycznych
Instrukcje tworzące zmienną lub redefiniujące jej wartość	.ASK .SETT .SETF	.ASKN .SETN .DEC .INC	.ASKS .SETS
Sposób zapisu stałej	—	oktalnie lub dziesiętnie z kropką	"TEKST"
Operatory działające na dany typ	—	+ — * / ! & = EQ <> NE > = GE < = LE > GT < LT	+ [od:do]

Uwagi: ! oznacza sumę logiczną
& oznacza iloczyn logiczny
[od:do] oznacza wycięcie tekstu

Tabela 2. Instrukcje we-wy i przypisania stosowane przy półautomatycznym generowaniu poleceń

Rodzaj instrukcji	Dozwolone postacie instrukcji	Uwagi
Wczytanie wartości logicznej	.ASK *ssssss tekst_zachęty*	Spodziewana jest odpowiedź jednoliterowa Y lub N. Nazwa zmiennej *ssssss* może mieć do sześciu liter.
Wczytanie wartości liczbowej	.ASKN *ssssss tekst_zachęty* .ASKN [*od:do*] *ssssss tekst_zachęty* .ASKN [*:std*] *ssssss tekst_zachęty* .ASKN [*od:do:std*] *ssssss tekst_zachęty*	Zmienne *od:do* stanowią dolne i górne ograniczenie spodziewanej wartości liczbowej, a *:std* wartość domyślną nadawaną przy wprowadzeniu odpowiedzi pustej.
Wczytanie wartości tekstowej	.ASKS *ssssss tekst_zachęty* .ASKS [*od:do*] *ssssss tekst_zachęty*	Zmienne *od:do* określają minimalną i maksymalną długość tekstu.
Przypisanie wartości logicznej	.SETT *ssssss* .SETF *ssssss*	Instrukcja .SETT przypisuje zmiennej *ssssss* wartość TRUE, a instrukcja .SETF wartość FALSE.
Przypisanie wartości liczbowej	.SETN *ssssss wyr_arytmetyczne*	Instrukcja przypisuje zmiennej *ssssss* wartość wyrażenia arytmetycznego.
Przypisanie wartości tekstowej	.SETS *ssssss wyr_tekstowe*	Instrukcja przypisuje zmiennej *ssssss* wartość wyrażenia tekstowego.

Tabela 3. Spis instrukcji interpretera plików poleceń

Skrócona postać zapisu	Objaśnienie
.etykieta:	Deklaracja etykiety, do której można przekazać sterowanie
.ASK .ASKN .ASKS	Instrukcje czytania z klawiatury dla typów: logicznego, liczbowego, tekstowego
.ERASE	Kasowanie zadeklarowanych symboli
.SETT lub .SETF .SETN .SETS	Instrukcje przypisania wartości: logicznych, liczbowych, tekstowych
.DATA .CLOSE .OPEN .OPENA	Instrukcje tworzenia plików: dodaj jedną linię na koniec pliku, zamknij plik, otwórz nowy plik, otwórz istniejący plik
.BEGIN lub .END .CHAIN .EXIT .GOSUB lub .RETURN .GOTO .ONERR .STOP	Instrukcje kontroli wykonania: otwórz lub zamknij blok interpretuj inny plik poleceń, opuść aktualny poziom, skocz do lub wróć z podprogramu, skocz, skocz po pojawieniu się błędu, zakończ interpretację, wykonaj inny plik poleceń jako podprogram
.IF .IFACT lub .IFNACT .IFDF lub .IFNDF .IFINS lub .IFNINS .IFLOA lub .IFNLOA .IFT lub .IFF .TEST	Instrukcje testowania: czy warunek logiczny jest spełniony, czy zadanie jest aktywne lub nieaktywne, czy symbol jest zdefiniowany lub niezdefiniowany, czy program jest zainstalowany lub niezainstalowany, czy program obsługi we-wy jest załadowany lub niezaładowany, czy zmienna logiczna ma wartość TRUE lub FALSE; instrukcja .TEST testuje wartość zmiennej tekstowej
.DISABLE lub .ENABLE	Wyłączanie lub włączanie takich opcji jak: makrogeneracja, wprowadzanie danych, globalność definicji, akceptowanie małych liter
.DEC lub .INC	Zmniejszanie lub zwiększanie wartości liczbowej
.DELAY .PAUSE .WAIT .XQT	Kontrola szybkości wykonania: zatrzymanie na określony czas, wstrzymanie, oczekiwanie na zakończenie programu, równoległe uruchomienie programu

zapisu parametrów ma postać ⟨nazwa_param⟩. Zestawienie znaczenia i właściwości niektórych parametrów zawarto w tabeli 4.

Tabela 4. Niektóre parametry interpretacji i ich znaczenie

Nazwa parametru	Typ wartości	Objaśnienie znaczenia parametru
⟨DEFAUL⟩	Logiczny	Parametr ma wartość TRUE, gdy po ostatnim pytaniu o liczbę wybrano wartość domyślną przez wprowadzenie odpowiedzi pustej (p. opis instrukcji .ASKN)
⟨MEMSIZ⟩	Liczbowy	Wielkość pamięci systemu w kilosłowach
⟨STRLEN⟩	Liczbowy	Długość tekstu w zmiennej po ostatnim wykonaniu instrukcji .ASKS lub .TEST
⟨DATE⟩	Tekstowy	Bieżący dzień, miesiąc i rok w formacie dd/mie/rr
⟨UIC⟩	Tekstowy	Identyfikator bieżącego katalogu w formacie [ggg,mmm]

Przykład wykorzystania plików poleceń

Na wydruku 1 przedstawiono przykład wzorca do generowania poleceń potrzebnych przy przetwarzaniu i uruchamianiu programu w języku Pascal. Polecenia te dotyczą edycji, translacji, konsolidacji, uruchomienia uzyskanego programu, a także kasowania zbędnych plików. Wydruk 2 zawiera przykład wykorzystania powyższego wzorca. Realizowana jest sekwencja następujących czynności: uruchomienie interpretera, wprowadzenie programu w języku Pascal, translacja pierwsza, sprawdzenie wydruku translatora, dokonanie poprawek, translacja druga, konsolidacja, wykonanie.

```
;PLIK POLECEN DO PRZETWARZANIA PROGRAMU W JEZ. PASCAL
.ASKS [1:9.] FILE NAZWA PROGRAMU (BEZ ROZSZERZENIA)
.ENABLE SUBSTITUTION
.1:
;ETAP WPROWADZANIA I POPRAWIANIA PROGRAMU
EDI 'FILE'.PAS
;ETAP TRANSLACJI
PAS 'FILE'.OBJ,'FILE'.LST/-U='FILE'.PAS
.ASK Y CZY POTRZEBUJESZ WYDRUK TRANSLATORA
.IFF Y .GOTO 2
PIP TI:='FILE'.LST
.2:
;USUWANIE ZBEDNYCH PLIKOW
PIP 'FILE'.LST;*/DE
PIP 'FILE'.PAS;*,'FILE'.OBJ;*/PU
.ASK Y CZY POJAWILY SIE BLEDY TRANSLACJI
.IFT Y .GOTO 3
;ETAP KONSOLIDACJI
TKB 'FILE'.TSK='FILE'.OBJ,LB:[1,1]PASLIB.OLB/LB
;USUWANIE ZBEDNYCH PLIKOW
PIP 'FILE'.*;*/PU
;ETAP TESTOWANIA DYNAMICZNEGO PROGRAMU
RUN 'FILE'.TSK/INC=5000
.WAIT
.3:
.ASK Y CZY ZAMIERZASZ POPRAWIC PROGRAM
.IFT Y .GOTO 1
.STOP
```

Wydruk 1

```
)@dopas
);PLIK POLECEN DO PRZETWARZANIA PROGRAMU W JEZ. PASCAL
)* NAZWA PROGRAMU (BEZ ROZSZERZENIA) [S R:1,-9.]:zadanie
);ETAP WPROWADZANIA I POPRAWIANIA PROGRAMU
)EDI zadanie.PAS
[CREATING NEW FILE]
INPUT
program zadanie(tty);
 begin
   writeln('zadanie dziala'))
 end.

 *ex
[EXIT]

);ETAP TRANSLACJI
)PAS zadanie.OBJ,zadanie.LST/-U=zadanie.PAS
PASCAL  PDP-11  VERSION  6.01
1985-11-07    17:16:24
ERROR IN LINE     3:    6
** ERROR DETECTED
TOTAL PROGRAM SIZE:          000126
OUTER MOST DATA SIZE:        000002
RESERVED STACK & HEAP SIZE:  001000
RUNTIME:        7  SECONDS
)* CZY POTRZEBUJESZ WYDRUK TRANSLATORA? [Y/N]:y
)PIP TI:=zadanie.LST

PASCAL  PDP-11  VERSION  6.01    1985-11-07   17:16:24
ZADANIE.PAS                                  .MAIN.

    1   PROGRAM ZADANIE(TTY);
    2     BEGIN
    3       WRITELN('ZADANIE DZIALA'))
*****                              ^6

    4   END.

***** ERROR DETECTED
TOTAL PROGRAM SIZE      000126
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   001000

PAS ZADANIE.OBJ,ZADANIE.LST/-U=ZADANIE.PAS
;USUWANIE ZBEDNYCH PLIKOW
)PIP zadanie.LST;*/DE
)PIP zadanie.PAS;*,zadanie.OBJ;*/PU
)* CZY POJAWILY SIE BLEDY TRANSLACJI? [Y/N]:y
)* CZY ZAMIERZASZ POPRAWIC PROGRAM? [Y/N]:y
)ETAP WPROWADZANIA I POPRAWIANIA PROGRAMU
)EDI zadanie.PAS
[00004 LINES READ IN]
[PAGE    1]
 *np3
   WRITELN('ZADANIE DZIALA'))
 *c/))/)/
   WRITELN('ZADANIE DZIALA')
 *ex
[EXIT]

)ETAP TRANSLACJI
)PAS zadanie.OBJ,zadanie.LST/-U=zadanie.PAS
PASCAL  PDP-11  VERSION  6.01
1985-11-07    17:17:27
NO ERROR DETECTED
TOTAL PROGRAM SIZE:          000126
OUTER MOST DATA SIZE:        000002
RESERVED STACK & HEAP SIZE:  001000
RUNTIME:        7  SECONDS
)* CZY POTRZEBUJESZ WYDRUK TRANSLATORA? [Y/N]:n
)USUWANIE ZBEDNYCH PLIKOW
)PIP zadanie.LST;*/DE
)PIP zadanie.PAS;*,zadanie.OBJ;*/PU
)* CZY POJAWILY SIE BLEDY TRANSLACJI? [Y/N]:n
)ETAP KONSOLIDACJI
)TKB zadanie.TSK=zadanie.OBJ,LB:[1,1]PASLIB.OLB/LB
)USUWANIE ZBEDNYCH PLIKOW
)PIP zadanie.*;*/PU
)ETAP TESTOWANIA DYNAMICZNEGO PROGRAMU
)RUN zadanie.TSK/INC=5000
ZADANIE DZIALA
)* CZY ZAMIERZASZ POPRAWIC PROGRAM? [Y/N]:n
)@ (EOF)
)
```

Wydruk 2

Zalety i ograniczenia półautomatycznego generowania poleceń

Mechanizm półautomatycznego generowania poleceń jest szczególnie przydatny w sytuacjach wielokrotnego powtarzania podobnych operacji przetwarzania. Pliki poleceń są z powodzeniem stosowane jako „żywy" fragment dokumentacji dostępnych handlowo generacyjnych pakietów oprogramowania.

Główne zalety stosowania plików poleceń są następujące:

- zmniejsza się ilość informacji wprowadzanej z klawiatury systemu
- zmniejsza się ryzyko wprowadzenia błędnej informacji
- system zachowuje się ponaglająco „pytając" o wartości konkretnych parametrów.

Dzięki tym zaletom realizacja danej sekwencji operacji jest mniej czasochłonna.

Stosowanie mechanizmu półautomatycznego generowania poleceń może okazać się niekorzystne w przypadku krytycznego obciążenia głównych zasobów systemu (pamięć i dyski) oraz wtedy, gdy procesor plików poleceń realizuje zbyt złożone funkcje obliczeniowe. Plik poleceń powinien być skonstruowany tak, aby błąd wykonania nie powodował utraty długotrwałego cyklu przetwarzania. Ze względu na sposób komunikowania błędów przez programy systemowe i użytkowe, mechanizm półautomatycznego generowania poleceń dla systemu RSX może jedynie wspomagać, a nie całkowicie zastępować, umiejętności osoby sterującej systemem.

Warto również podkreślić, że wykorzystanie poprawnie skonstruowanych plików poleceń ułatwia i przyspiesza przetwarzanie, co w konsekwencji powoduje zmniejszenie kosztów przetwarzania.

LITERATURA

[1] Digital Equipment Corporation: RSX-11M/M-PLUS MCR Operations Manual. Maynard (MA), 1979

[2] Digital Equipment Corporation: RSX-11 Utilities Procedures Manual. Maynard (MA), 1977

[3] Ericsson: Pascal User Manual. Stockholm, 1979.

ANDRZEJ MACIOŁ
ADAM STAWOWY
Instytut Organizacji i Zarządzania Przemysłem
Akademia Górniczo-Hutnicza
Kraków

Środki do przetwarzania tekstów — historia i dzień dzisiejszy

Głównym celem działalności kadry kierowniczej przedsiębiorstw było zawsze zwiększenie produktywności. Przez długi czas koncentrowano się na wzroście wydajności pracowników bezpośrednio produkcyjnych. Jednak w ostatnich latach, w miarę zbliżania się do kresu możliwości organizacyjnych i technologicznych wzrostu wydajności w wydziałach fabrycznych, poczęto kłaść coraz większy nacisk na produktywność pracowników umysłowych. Szybko uświadomiono sobie, iż otoczenie biurowe stwarza duże możliwości poprawy jakości i szybkości pracy. Szczególnie ostro uwidocznił się ten problem w warunkach amerykańskich, gdzie już w 1972 roku wytwarzaniem usług zajmowało się ponad 60% prywatnych firm, w których — siłą rzeczy — praca biurowo-administracyjna jest najistotniejszym elementem [1].

Rozwiązaniem tego problemu okazała się automatyzacja niektórych prac biurowych. Właśnie wtedy, w 1969 roku, Ulrich Steinhilper — przedstawiciel firmy IBM na RFN — stworzył termin textverarbeitung, przełumaczony później na język angielski jako word processing. Pojęcie word processing przetwarzanie tekstów jest różnie rozumiane. Często cytowana definicja ANSI brzmi następująco [2]: *Przetwarzanie tekstów tworzy system złożony z ludzi, procedur i wyposażenia, który transformuje idee i informacje w pisany komunikat; system taki powstaje wskutek zaprojektowania, wdrożenia i zarządzania zasobami pracy biurowej w celu osiągnięcia maksymalnej wydajności pracy administracji.*

Jest to pojęcie bardzo szerokie, obejmujące swym zakresem:

- wyrażanie i przekazywanie informacji w celu przełożenia ich na użyteczną formę,
- tworzenie dokumentów zawierających te informacje,
- kopiowanie dokumentów,
- archiwowanie ich w celu ewentualnego późniejszego wykorzystania,
- rozprowadzanie dokumentów do odbiorców.

Rzecz jasna, iż trudno byłoby przedstawić wszystkie aspekty tak rozumianego przetwarzania tekstów, dlatego w tym artykule zajęto się newralgicznym punktem tego procesu, tj. tworzeniem dokumentów przez pisanie na maszynie bądź drukowanie. Celem artykułu jest więc przedstawienie nowoczesnych urządzeń do przetwarzania tekstów oraz możliwości, jakie one stwarzają.

Według normy DIN 002140 urządzenie do przetwarzania tekstów (niem. Textautomat) jest to [6] *maszyna biurowa do przyjmowania i emisji tekstów oraz ich przetwarzania, która posiada co najmniej następujące urządzenia:*

- *urządzenie do wprowadzania tekstów,*
- *jednostkę centralną, która przy użyciu programu maszynowego może prowadzić w jednym przebiegu obróbkę tekstu o wielkości przekraczającej jedną stronę formatu A-4,*
- *nośnik tekstu do zapamiętywania co najmniej jednej strony formatu A-4,*
- *urządzenie do sterowanej automatycznie emisji dokumentów.*

Wymienione warunki są spełnione przez scharakteryzowane poniżej grupy urządzeń elektronicznych.

PODZIAŁ I CHARAKTERYSTYKA ŚRODKÓW DO PRZETWARZANIA TEKSTÓW

Procesory tekstów z reguły dzieli się na 4 główne grupy [2]:

- **elektroniczne (inteligentne) maszyny do pisania (ang. inteligent typewriters),**
- **procesory samodzielne (ang. stand-alone text editors),**
- **systemy współdzielone (ang. shared-logics systems),**
- **duże komputery.**

Elektroniczne maszyny do pisania

Historia maszyny do pisania liczy już blisko 120 lat. Mianowicie, w 1869 r. została opatentowana przez dziennikarza

Dr inż. ANDRZEJ MACIOŁ ukończył w 1979 roku studia na Wydziale Organizacji i Zarządzania Przemysłem Akademii Górniczo-Hutniczej, a w 1983 roku uzyskał tytuł doktora nauk ekonomicznych (praca doktorska z dziedziny symulacji procesów zarządzania) w Politechnice Wrocławskiej. Pracuje w Instytucie Organizacji i Zarządzania AGH na stanowisku adiunkta. Zajmuje się projektowaniem informatycznych systemów zarządzania.

Mgr inż. ADAM STAWOWY ukończył w 1980 roku studia na Wydziale Organizacji i Zarządzania Przemysłem Akademii Górniczo-Hutniczej. Pracuje w Instytucie Organizacji i Zarządzania AGH na stanowisku starszego asystenta. Zajmuje się głównie zastosowaniem mikrokomputerów w zarządzaniu.

amerykańskiego C. L. Sholesa pierwsza użyteczna maszyna do pisania, którą 4 lata później wprowadzono na rynek. Już podczas pierwszej wojny światowej pojawiły się automatyczne maszyny do pisania, służące do tworzenia powtarzalnych listów (smutna wojenna konieczność) przy użyciu perforowanej wstęgi papieru. Druga wojna światowa przyniosła następne zmiany — mechaniczną maszynę ze wstęgą papierową wyparła elektryczna z tasiemką perforowaną. W latach pięćdziesiątych pojawiła się maszyna (Flexowriter firmy Friden) zdolna do komponowania dokumentów przy użyciu dwóch lub większej liczby tasiemek. Te automaty nie pozwalały jeszcze na korektę ewentualnych błędów, ani na obróbkę tekstów w dzisiejszym znaczeniu.

Przełom nastąpił w 1964 roku, kiedy firma IBM zaprezentowała swoją MT/ST (Magnetic Tape Selectric Typewriter), która umożliwiała zapamiętywanie tekstu na taśmie magnetycznej. I choć stało się możliwe poprawianie błędnie napisanych słów, usuwanie zbędnych czy dopisywanie nowych, trzeba było sporych starań reklamowych IBM, by wreszcie MT/ST odniosła rynkowy sukces. Początkowe trudności wynikły z wysokiej ceny i wcale nieprostej obsługi nowego produktu. Niedługo później do MT/ /ST dołączono drugą jednostkę pamięci taśmowej, co znacznie rozszerzyło możliwości maszyny — przepisywanie z taśmy na taśmę, tworzenie dokumentów z dwóch taśm, automatyczne formatowanie itp.

W 1969 roku firma IBM wprowadziła na rynek MC/ST (Magnetic Card Selectric Typewriter), w której nośnikiem informacji była karta magnetyczna odzwierciedlająca jedną stronę tekstu (50 wierszy × 100 kolumn). Uprościło to znacznie obsługę maszyny, przy zachowaniu wszystkich zalet jej poprzedniczki.

W 1972 roku pojawiła się pierwsza elektroniczna maszyna do pisania z pamięcią wewnętrzną.

Elektroniczne maszyny do pisania przypominają zewnętrznie znane powszechnie maszyny elektromechaniczne (elektryczne). Dzięki zastosowaniu techniki mikroprocesorowej możliwe jest wyposażenie tych urządzeń w funkcje przypisywane automatom do przetwarzania tekstów. Już w urządzeniach średniej klasy istnieje możliwość wykrywania i usuwania błędów w pamięci. Słowa o trudnej pisowni lub często używane mogą być przechowywane na stałe. Pamięć główna lub robocza umożliwia wielokrotne powielanie tekstów w całości lub fragmentach. Bardzo pożyteczne są również urządzenia do automatycznego ustawiania marginesu i blokowania zdań.

Maszyny do pisania wysokiej klasy dają dodatkowo możliwość komponowania tekstów. Przykładowo: list jednakowej treści można zaopatrzyć różnymi adresami odczytanymi z pamięci dyskowej (najczęściej są to dyski elastyczne). Szybkość pisania tych urządzeń jest duża, co ma istotne znaczenie przy emisji znacznej liczby dokumentów. Łatwe jest dokonywanie poprawek. Możliwe jest — na przykład — przeskakiwanie do dowolnych fragmentów i znaków już napisanego tekstu. Często urządzenia te są wyposażone w możliwość dokonywania prostych obliczeń (zazwyczaj w zakresie czterech podstawowych działań). Stosuje się także automatyczne dzielenie zgłosek, jako działanie autonomiczne maszyny lub w formie propozycji do zaakceptowania przez operatora. W przeważającej większości rozwiązań stosuje się wymienne głowice lub tarcze, zawierające do stu różnych znaków.

Najnowocześniejsze maszyny do pisania mogą być w sposób czynny dołączone do sieci komputerowej lub telefonicznej. Sieć telefoniczna umożliwia współpracę z podobnymi urządzeniami tej samej klasy lub z komputerami.

Rynek maszyn do pisania z pamięcią jest podzielony między wielu wytwórców. W Europie przodującą rolę odgrywają firmy: Olivetti, Triumph-Adler i Olympia, które np. w RFN opanowały, jak się ocenia, 80% rynku. Firma IBM, która zawładnęła rynkiem elektronicznych maszyn do pisania w 50%, w zakresie maszyn z pamięcią nie odgrywa znaczącej roli. Oprócz wymienionych, do znaczących producentów w tej dziedzinie można zaliczyć firmy: Brother, Hermes, Facit, Xerox i Sharp. W ostatnim roku na rynki zachodniej Europy wchodzą wyroby firm Canon, Exxon, SCM Smith-Corona, Silver Reed, a także Utax ze Szwajcarii i — co ciekawsze — Robotrona z NRD. W samej RFN w 1982 roku wyprodukowano 600 000 elektrycznych i elektronicznych maszyn do pisania. Jedną trzecią tej liczby stanowią maszyny do pisania z pamięcią. W całej Europie w 1983 roku — jak wykazały badania firmy Ericson — było zainstalowanych 9,5 mln maszyn do pisania, w tym 79% ręcznych i elektrycznych, a 17% maszyn z pamięcią. Pozostałe 4% stanowiły systemy przetwarzania tekstów.

Jako przykład najnowszych osiągnięć producentów maszyn z pamięcią można wymienić ofertę firmy Olivetti. Obejmuje ona dwie grupy urządzeń: małe maszyny z serii Praxie i duże urządzenia serii ET. Urządzenia serii ET można zaliczyć do „klasy komfortowej". Model ET 115 ma stałą pamięć o pojemności 1830 znaków, służącą do zapamiętywania fragmentów tekstów i formularzy. Display przedstawia 24 znaki, w tym 17 znaków tekstu. Pozostałe znaki przedstawiają stan pamięci urządzenia.

Maszyna ET 225 ma dodatkowo kalendarz i zegar. Dzięki dołączonym modułom, zdolność zapamiętywania znaków można poszerzyć do 64 000 znaków. Model ET 351 ma zewnętrzną pamięć na dyskietce. Można w niej zapisać 128 000 znaków. Model ma również 40-znakowy wyświetlacz (ang. display) i charakteryzuje się zwiększoną szybkością pisania oraz możliwością automatycznej wymiany arkuszy. Każdy z opisanych modeli serii ET ma 500-znakową pamięć poprawek. Firma Olivetti oferuje także około 20 wymiennych głowic do pisania.

Dwoma najnowszymi modelami firmy Olympia są ES 180 Disque i Supertype 2. W bieżącym roku firma ma zamiar wprowadzić na rynek jeszcze nowszy model ES 340. Wersję Supertype 2 określono jako elektroniczną maszynę z pamięcią do wyposażenia stanowisk pracy biurowej o wysokich wymaganiach. Charakteryzuje ją programowalny wysuw papieru (do 25 wierszy), urządzenie wprowadzania poprawek z 500-znakową pamięcią, cztery specjalne sposoby drukowania (kursywa, tłusty druk itp.), druk podwójny i automatyczne podkreślanie. Całość kontrolowana jest przez wyświetlacz o 20 wierszach.

Model Disque posiada dodatkowo 48 adresowalnych segmentów pamięciowych. Pozwalają one na komponowanie fragmentów tekstów. Do jego wyposażenia należą także: pamięć tekstu (16 000 znaków), minidyskietka (8000 znaków), program rozdzielania zgłosek, 40-znakowy wyświetlacz oraz sprzęg V.24. Model ten został dopuszczony w RFN do stosowania jako pocztowe urządzenie dalekopisowe.

Model ES 180 może przetwarzać bloki zdań, ma rozdzieloną pamięć stałą i roboczą, automatyczne rozdzielanie zgłosek i prowadzenie pojedynczych arkuszy (do 250).

Podobnie bogatą ofertę przedstawia wiele innych firm walczących o pozycję na rynku producentów maszyn do pisania.

Procesory samodzielne

Procesory samodzielne mają najlepsze cechy elektronicznych maszyn do pisania, umożliwiając wyświetlanie pisanego tekstu na ekranie monitora, a drukowanie przenosząc na oddzielną, przyłączoną drukarkę. Początkowo stosowane procesory Wang 10 A czy Vydec 1200 [2] są wypierane teraz przez różnego rodzaju mikrokomputery. Choć historia mikrokomputera liczy zaledwie 8 lat, już szacuje się, że przetwarzanie tekstów zajmuje najwięcej czasu pracy mikrokomputerów [7]. Przyczyna jest prosta — korzystanie z jednostki ekranowej umożliwia dokonywanie wszystkich zmian w tekście z dużą łatwością. Ponadto teksty można przetwarzać w domu, a ich przechowywanie i przenoszenie na dyskietce nie przedstawia większych trudności. Użytkownik musi jedynie „wystukać" tekst na klawiaturze; cała reszta — związana z układem tekstu, dzieleniem go na wiersze, akapity itp. — jest robiona automatycznie. Papier potrzebny jest tylko do wydrukowania ostatecznej wersji, gdyż podczas przetwarzania jest ona widoczna na ekranie.

W skład typowego procesora tekstowego wchodzą następujące urządzenia:

a) jednostka centralna z mikroprocesorem, pamięcią ROM i RAM oraz sprzęgami urządzeń zewnętrznych,

b) klawiatura alfanumeryczna,
c) jedna lub dwie stacje dyskietek często wbudowane w jednostkę centralną,

d) monitor ekranowy,
e) drukarka.

Procesor tekstowy, czyli mikrokomputer do przetwarzania tekstów musi ponadto spełniać kilka podstawowych warunków [7]:

- klawiatura musi być wygodna dla użytkownika oraz zawierać wszystkie znaki używane w przetwarzaniu tekstów odpowiednio rozmieszczone; powszechnie stosuje się układ QWERTY, chociaż czynione są próby z innymi układami o większej wydajności,

- zamiast monitora, do mniej poważnych prac można używać telewizora, ale w zastosowaniach profesjonalnych trzeba stosować oddzielne monitory wysokiej klasy, wyświetlające minimum 80 znaków w 20 wierszach; nie muszą to być monitory kolorowe, gdyż wystarczą monochromatyczne, ale o barwach nie męczących oczu (zielone lub żółte litery na czarnym tle),

- wśród urządzeń pamięci zewnętrznej standardem są stacje dyskietek 5 1/4 cala o pojemności 0,5—2 MB lub stacje mikrodyskietek 3 1/2 cala (3 cale) o pojemności 0,3—0,8 MB; wykorzystywane są również jednostki szybkiej pamięci taśmowej i stacje dysków sztywnych typu Winchester,

- użyty mikroprocesor może być 8-, 16- lub 32-bitowy, ale pojemność pamięci wewnętrznej RAM musi wynosić minimum 64 KB, a nawet 128 KB i więcej,

- najbardziej typowym systemem operacyjnym jest CP/M i jego odpowiedniki, ze względu na dużą liczbę programów pracujących pod jego nadzorem; z liczących się wytwórców, jedynie firma Apple oferuje swój własny system operacyjny niekompatybilny z CP/M, ale może sobie na to pozwolić, ze względu na mocną pozycję na rynku.

Dobrym przykładem możliwości procesorów samodzielnych jest mikrokomputer Macintosh firmy Apple. Oparta na 16-bitowym mikroprocesorze Motorola 68000 jednostka centralna ma wbudowaną pojedynczą stację mikrodyskietek 3 1/2 cala o pojemności 400 KB oraz wbudowany monitor monochromatyczny o przekątnej 9 cali, wyświetlający 80 znaków w 25 wierszach. Mikroprocesor adresuje pamięć wewnętrzną RAM o pojemności 128 KB i 60 KB ROM. Dołączona klawiatura ma 38 przycisków w układzie QWERTY: do dokonywania wyboru z wykazu dostępnych opcji (ang. menu) służy myszka.

W skład oprogramowania dostarczanego przez producenta wchodzi m.in. MacWrite, czyli pakiet programów do przetwarzania tekstów. MacWrite jest zbudowany hierarchicznie i daje użytkownikowi następujące możliwości [4]:
— składanie dokumentu z dowolnej liczby uprzednio zapamiętanych,
— dzielenie dokumentów na kilka nowych,
— wstawianie poprawek do tekstu,
— wymazywanie słów, zdań i większych fragmentów tekstu,
— kopiowanie dokumentów,
— przesuwanie określonych fragmentów,
— poszukiwanie określonych fragmentów,
— zamiana fragmentów między dokumentami,
— drukowanie dokumentów,
— składowanie, archiwowanie dokumentów,
— powrót do stanu sprzed wykonania ostatniej instrukcji,
— pisanie 15 rodzajami czcionki; każda czcionka może mieć 6 wzorów pisma i 9 rozmiarów; tekst może być prosty bądź pochylony w lewo lub w prawo,
— automatyczne ustawianie akapitu, lewego i prawego marginesu, odstępów między wierszami, standardowej i dziesiętnej tabulacji,
— automatyczne dosuwanie tekstu,
— wybór kierunku przesuwania przeglądania tekstu w górę, w dół, w lewo lub w prawo.

Aby wykorzystać wymienione i inne możliwości, należy do systemu dołączyć odpowiednią drukarkę. Mikrokomputer ma tę przewagę nad elektroniczną maszyną do pisania, że w czasie nie wykorzystywanym na przetwarzanie tekstów można go łatwo wykorzystać do przetwarzania danych. Tak więc, mikrokomputer odpowiednio oprogramowany rozwiązuje większość problemów powstałych we współczesnym biurze, w pracach administracyjnych i kierowniczych.

Systemy współdzielone

Systemy współdzielone są oparte na koncepcji centralizacji pewnych funkcji i zasobów oraz udostępnieniu ich wszystkim stanowiskom roboczym. Do centralnych zasobów należą najczęściej duże zbiory danych na dyskach sztywnych, urządzenia teletransmisyjne, szybkie drukarki, jednostki przetwarzania lub czytniki optyczne [3].

Współdzielenie powoduje komplikacje w oprogramowaniu. Centralny system operacyjny musi być oprogramowany w ten sposób, aby istniała możliwość tworzenia kolejek zadań, dynamicznego przydzielania zasobów i obsługi przerwań. Z reguły są to systemy wieloterminalowe, oparte na minikomputerze, pracujące w czasie rzeczywistym. Podział cech między jednostką centralną a stanowiskami roboczymi musi być dobrze przemyślany, choć w miarę rozwoju techniki mikroprocesorowej coraz większa część pracy związana z manipulowaniem tekstami jest wykonywana na stanowisku roboczym. Tak więc, tekst jest wprowadzany i redagowany na stanowisku roboczym, zapisywany na dyskietce, a dokumenty mogą być drukowane na lokalnej drukarce. Większe wydruki są wykonywane na centralnej, szybkiej drukarce, a do dłuższego przechowywania — dokument może być przesłany przy użyciu prostej instrukcji do centralnej stacji sztywnych dysków. Inteligentne stanowisko robocze pozwala operatorowi wykonywać dużą część pracy, nawet w tym wypadku, gdy jednostka centralna ulegnie awarii.

Ponieważ systemy współdzielone są kosztowne, muszą być starannie zaprojektowane, ich zakup uzasadniony ekonomicznie, a eksploatacja — sprawnie zarządzana. Konieczne jest zaprojektowanie dużej bazy danych (niejednokrotnie na kilkanaście tysięcy dokumentów), rozwiązanie złożonych problemów telekomunikacji i — odpowiednia konserwacja systemu.

Pojawiają się też trudności natury organizacyjno-psychologicznej. Chociaż centralna baza danych jest rozwiązaniem bardziej efektywnym niż lokalne przechowywanie danych, użytkownicy niechętnie godzą się na rezygnację z kontroli nad swoimi zbiorami i powszechny dostęp do nich. W rezultacie, niezbędne jest wprowadzenie ochrony zasobów i ograniczenie dostępu do nich, co powoduje pewne niedogodności w codziennym użytkowaniu.

Na rynku jest dostępnych wiele systemów współdzielonych, różniących się stopniem podziału zadań między jednostkę centralną a stanowiskami roboczymi, liczbą stanowisk oraz pojemnością pamięci dyskowej. Jednym z pierwszych była seria Wang OIS/100 (Office Information System). W skład największego z tej serii systemu 140 wchodzą:

- jednostka centralna oparta na mikroprocesorze, z pamięcią wewnętrzną RAM 64 KB i stacji dysków sztywnych o pojemności 100 MB,

- 24 inteligentne stanowiska robocze, każde z mikroprocesorem i pamięcią RAM 32 KB,

- 8 inteligentnych drukarek, mikroprocesor z pamięcią RAM 16 KB,

- optyczny czytnik dokumentów OCR jako urządzenie wejściowe.

System ma zastosowanie w wielkich ośrodkach przetwarzania tekstów i w biurach wprowadzających pełną automatyzację prac biurowych, aczkolwiek istnieje możliwość przetwarzania rozproszonego (ang. distributed processing), gdyż każde stanowisko może być połączone kablem z jednostką centralną na odległość nie przekraczającą 700 m. System jest szczególnie wydajny przy sortowaniu (porządkowaniu) dużych zbiorów tekstowych.

Przetwarzanie tekstów na dużych komputerach

Wbrew pozorom przystosowanie dużego komputera do prac nad przetwarzaniem tekstu nie jest łatwe. Co prawda, problemy teletransmisji i dołączenia dużej liczby terminali czy też zapewnienie podstawowych wymagań stawianych procesorom nie są trudne do rozwiązania. Duże komputery dysponują też masową pamięcią dyskową, szybkimi drukarkami, a programy dostępu do baz danych są dobrze znane.

Podstawową trudnością w wykorzystaniu dużego komputera do przetwarzania tekstów jest tzw. problem sprzężenia użytkowego (ang. user interface), czyli sposób ko-

munikowania się użytkownika z komputerem i korzystania z zasobów systemu. Komunikowanie się z komputerem podczas przetwarzania tekstów nie może być skomplikowane. Użytkownik musi radzić sobie bez gruntownego przeszkolenia i znajomości systemu operacyjnego maszyny. Musi więc istnieć dobrze zbudowane sprzężenie programistyczne, które potrafi przetłumaczyć żądania użytkownika na język maszyny.

Wyróżnia się trzy stosowane konfiguracje sprzętowe dużych komputerów:

- Pierwsza — obejmuje system z podziałem czasu i nieinteligentne terminale. Całe oprogramowanie i zasoby są scentralizowane, a całość przetwarzania tekstów przejmuje główny komputer. Ten sposób korzystania z dużego komputera jest bardzo nieefektywny i skomplikowany.

- Drugi rodzaj konfiguracji, stanowiący system z podziałem czasu i dołączonymi inteligentnymi terminalami, odpowiada poglądowi, iż lokalne stanowisko robocze musi dysponować własnym sterowaniem i minimalną pamięcią masową. Natomiast do jednostki centralnej dołączone są: czytnik optyczny OCR, szybka drukarka i stacje dysków o dużej pojemności. Na pierwszy rzut oka opisana konfiguracja jest podobna do systemów współdzielonych. O ile jednak te drugie służą tylko do przetwarzania tekstów i prostych obliczeń, w systemach z podziałem czasu przetwarzanie tekstów jest jednym z zadań i to — nie najważniejszym. Użytkownik musi najpierw wprowadzić do pamięci operacyjnej pakiet programowy służący do przetwarzania tekstów, co wymaga użycia skomplikowanego języka poleceń. Oczywiście im więcej funkcji można wykonywać na stanowisku roboczym, tym prostsze jest posługiwanie się systemem z podziałem czasu. W przeciwnym razie użytkownik musi przejść gruntowne przeszkolenie (co przy kilkudziesięciu stanowiskach jest kosztowne i skomplikowane) albo korzystać z pośrednictwa zawodowego operatora.

- W systemach o przetwarzaniu rozproszonym (ang. distributed processing systems) główne funkcje przetwarzania tekstów przejmuje procesor zewnętrzny i dołączone do niego inteligentne terminale. Procesor komunikuje się z komputerem macierzystym tylko w szczególnych wypadkach. Przykładem takiego podejścia jest system 3730 firmy IBM, przedstawiony na przełomie lat 1978/1979 [2].

System ten składa się z trzech części i współpracuje z komputerem macierzystym IBM 370. Tekst jest wprowadzany przez inteligentne terminale 3732, które są przystosowane do wykonywania kilku funkcji przetwarzania tekstu jak: wstawianie nowych fragmentów, wymazywanie, poprawianie znaków i automatyczne ustawianie marginesu. Do wyświetlania tekstu służy ekran o wielkości 24 linie po 80 znaków. Drugą, podstawową częścią systemu 3730 jest sterownik (procesor) typu 3791. Ma on za zadanie obsługę 3732 stanowisk roboczych, 3736 (do 16 sztuk) drukarek oraz komunikowanie się z komputerem macierzystym. Każdy sterownik jest wyposażony w pamięć na dyskach sztywnych (10—33 MB), pamięci na dyskach elastycznych i na kartach magnetycznych. Funkcje przetwarzania tekstów nie realizowane przez stanowisko robocze mogą być wykonywane przez sterownik. System 3730 ma ograniczone możliwości, np. nie pozwala na łatwą generację powtarzalnych dokumentów, nie przeprowadza kontroli pisowni, brak mu również możliwości automatycznego wprowadzania szczególnie często powtarzających się wyrazów. Dopiero jego dołączenie do komputera IBM 370 przez specjalną sieć (SNA — Systems Network Architecture) umożliwia wykonywanie takich funkcji jak:

— wykorzystanie elektronicznej poczty (ang. electronic mail) do komunikowania się z wieloma użytkownikami

— korzystanie z wielkich baz danych komputera 370

— korzystanie ze zbiorów i możliwości MIS (Management Information Systems — systemy informowania kierownictwa)

— eliminacja wprowadzania danych już wprowadzonych do systemu w innych przebiegach (nie — w czasie przetwarzania tekstów)

— automatyczne dzielenie wyrazów

— prowadzenie kontroli pisowni

— dokonywanie obliczeń oraz zestawień statystycznych i umieszczanie ich w tekście

— tworzenie zbiorów wejściowych dla fotoskładu.

OCENA POSZCZEGÓLNYCH GRUP PROCESORÓW TEKSTÓW

Właściwa ocena możliwości poszczególnych grup procesorów tekstów zależy od prawidłowego doboru środków przetwarzania tekstów w pracy biurowej. W tabeli 1 przedstawiono zalety i wady poszczególnych typów systemów w taki sposób, że charakterystyka jednego systemu jest odnoszona do opisu jego poprzednika w tabeli.

Tabela 1. Zalety i wady poszczególnych grup procesorów tekstów [2]

Typ systemu	Zalety	Wady
Inteligentne maszyny do pisania	Stosunkowo tanie, łatwe do opanowania przez obsługę, mające pewne możliwości komunikacyjne, mieszczące się na biurku.	Ograniczona pamięć, ograniczone możliwości, niekompatybilne między sobą.
Procesory samodzielne	Łatwe do opanowania, łatwe w obsłudze, szybkie w druku dzięki dołączonym drukarkom, kompatybilne między sobą (dzięki wspólnym systemom operacyjnym), mające możliwość komunikowania się z innymi użytkownikami, zapewniające wiele opcji.	Droższe, zajmujące więcej miejsca, wymagające zmian organizacyjnych w biurze.
Systemy współdzielone	Większa produktywność, niższe koszty eksploatacji, większe możliwości, możliwość podziału skomplikowanych zadań	Wysokie koszty zakupu, potrzeba zarządzania środkami i istnienia dobrze opracowanych procedur, skomplikowana obsługa, problemy ochrony zasobów.
Duże komputery systemy z podziałem czasu	Duża różnorodność funkcji, duża pamięć operacyjna i masowa, niskie koszty zakupu (tylko terminale).	Wymaganie dużej wprawy użytkowników, duże koszty eksploatacyjne pamięć i czas komputera głównego, potrzeba dobrze opracowanych procedur.
Duże komputery systemy o przetwarzaniu rozproszonym	Duża pamięć operacyjna i masowa, bardziej elastyczne w działaniu, duża różnorodność funkcji.	Większe koszty zakupu, skomplikowane problemy połączeń.

Pożądane dziedziny zastosowania poszczególnych grup procesorów tekstu przedstawia tabela 2.

PRZETWARZANIE TEKSTÓW W POLSCE

W polskiej literaturze fachowej brak jest właściwie odpowiednika pojęcia „word processing". Nie jest to problem tylko lingwistyczny, ale raczej konsekwencja bardzo słabego rozwoju tej dziedziny przetwarzania informacji w kraju. Niemal wszystkie polskie podręczniki z dziedziny informatyki pomijają zagadnienie przetwarzania tekstów. Również w praktyce niewiele można znaleźć przykładów skutecznego wykorzystania urządzeń automatycznych do emisji dokumentacji. Do niedawna praktycznie niedostępne były nowoczesne urządzenia do przetwarzania tekstów. Wyjątek stanowiły automaty organizacyjne Optima (produkcji NRD), rozprowadzone przez Predom-Org na początku lat siedemdziesiątych. Urządzenia te jednak nie znalazły szerszego zastosowania, z powodu dużej awaryjności, niedogodności związanych z zastosowaniem taśmy papierowej jako nośnika informacji i słabego oprogramowania. Znanych jest zaledwie kilka publikacji o praktycznym wykorzystaniu automatów [5]. Mimo szybkiego rozwoju elektroniki, odczuwalnego również w naszym kraju, sytuacja w dziedzinie przetwarzania tekstów uległa tylko nieznacznej poprawie. W Polsce obecnie są dostępne następujące urządzenia, które można wykorzystać do przetwarzania tekstów:

- elektroniczne maszyny do pisania firmy Robotron S6010,

Tabela 2. Zastosowanie procesorów tekstów [2]

Typ systemu	Rodzaje dokumentów i ich cechy
Inteligentne maszyny do pisania	Codzienna korespondencja, pisanie pisanie do 20 powtarzalnych listów (ograniczenie ze względu na pamięć), dokumenty o krótkim okresie przechowywania, mała liczba pamiętanych paragrafów i formatów, pojedyncze dokumenty (rzadko stosowane), adresowanie kopert, kart itp.
Procesory samodzielne	Codzienna korespondencja, powtarzalne listy, dokumenty o zróżnicowanym okresie przechowywania, dokumenty z różną ilością sprawdzanych informacji, dokumenty z dużą ilością informacji numerycznych, większa liczba pamiętanych paragrafów, formularze, zbiory wejściowe do fotoskładu, teksty.
Systemy współdzielone	Dokumenty o zróżnicowanym okresie przechowywania, dystrybucja długich dokumentów, dokumenty z różną ilością sprawdzanych informacji, formularze i rubryki, duża liczba pamiętanych paragrafów, teksty, powtarzalne listy do wielu adresatów.
Duże komputery	Długie dokumenty (tekstowe lub numeryczne), zbiory wejściowe do przetwarzania danych, zbiory wejściowe do fotoskładu, powtarzalne listy do wielu adresatów (nazwiska posortowane według listy).

S6011 sprzedawane w cenie około 150—200 tys. zł przez Predom-Org,

- tzw. komputery biurowe, m.in. firmy Robotron A5110, A5120, A5130, Robotron 1715 oraz Elwro 514 produkcji zakładów we Wrocławiu,
- mikrokomputery konstruowane w firmach polonijnych i państwowych, np. Mera 60 (Mera Ster), MK-45 (KFAP), IMP-86 (IMPOL) itd., chociaż dostawcy nie zawsze dysponują odpowiednim oprogramowaniem i odpowiednimi drukarkami (na ogół drukarki typu DZM do tych celów się nie nadają).

Oferta krajowa jest więc dość uboga. Jednocześnie, rozpowszechnione w Polsce duże komputery (Odra i Riad) zupełnie się nie nadają do przetwarzania tekstów, ze względu na brak odpowiednich urządzeń zewnętrznych. Jeszcze skromniejsza jest oferta firm programistycznych, które można policzyć na palcach. Praktycznie tylko Computer Studio Kajkowscy z Gdyni oferuje system tekstowy, który może być wykorzystany na większości dostępnych mikrokomputerów. Niektórzy producenci sprzętu oferują również systemy tekstowe opracowane dla własnych urządzeń (np. Impol, Robotron czy Mera-Elwro).

Przedstawiona sytuacja wynika — jak się wydaje — nie tyle z niedostatków sprzętowych, co z nikłego zainteresowania problematyką przetwarzania tekstów. Warto więc pamiętać, że proste urządzenia do automatycznego przetwarzania tekstów mogą być w wielu wypadkach alternatywą dla kosztownych systemów przetwarzania danych [6]. W sytuacji, kiedy tylko bogate firmy stać jest na zakup drogich i trudno dostępnych komputerów, należy dążyć do spopularyzowania tej mało znanej w kraju dziedziny informatyki.

LITERATURA

[1] Kearny A. T. and Company Inc.: Managing in a „Service" Economy — A Research Study on Improving Clerical Productivity. Chicago, 1972

[2] Cecil P. B.: Word processing in the modern office. 1980, Menlo Park, Benjamin/Cummings

[3] Europejski Program Badawczy Diebolda. Narzędzia i systemy automatyzacji biura. Warszawa, 1982

[4] Miller M. J.: Introducing Lisa-like technology in a personal package. Popular Computing, Vol. 3, No. 5, 1984

[5] Ogólnopolska Konferencja NOT. Środki organizacyjno-techniczne warunkiem wdrażania nowoczesnej organizacji, Warszawa, 1980

[6] Textverarbeitung und Informatik-Fachtagung der GI. Bayreuth, Mai 1980. Informatik-Fachberichte, Springer-Verlag, Berlin, 1980

[7] Word processing — selected hardware. Computer and Programs. Vol. 1, No. 1, 1984.

Gdańsk

mikrokomputery

doradztwo

ODiTK

może poprawić

trafność decyzji

EO/524/K/86

Mikrokomputery w Wielkiej Brytanii

Pierwsze mikrokomputery w Wielkiej Brytanii — Altair, South West Technical Products i Sol — pojawiły się w 1977 roku, tj. trzy lata później niż w Stanach Zjednoczonych. Rok 1978 przyniósł wprowadzenie mikrokomputerów PET, Apple II, TRS-80 Model 1 i Exidy Sorcerer, ale ich ceny skutecznie odstraszały przeciętnych hobbistów. Ceny początkowe sprzętu z 1978 roku odegrały dużą rolę w podziale rynku. Wysoka cena Apple'a II, ponad 1000 funtów, spowodowała, że dużo tańszy PET zdominował rynki Wielkiej Brytanii, RFN i Skandynawii. Pozwoliło to przetrwać firmie Commodore lata dominacji Apple'a w USA.

Prawdziwi brytyjscy miłośnicy mikrokomputerów potrzebowali jednak czegoś jeszcze tańszego. Odzew ze strony przemysłu przyszedł szybko w postaci dostarczania komputerów w częściach do samodzielnego składania, jak na przykład: Nascom 1, Mk14 (Science of Cambridge) i późniejszy UK101, sprzedawany poniżej 200 funtów. Były to jednopłytkowe (ang. single board) komputery, przeważnie o pamięci RAM od 2 do 4 KB, z szesnastkową klawiaturą gumową (klawiatura maszynowa ze znakami ASCII, w jaką wyposażono Nascom 1 była prawdziwym luksusem) i z programem monitora w kodzie maszynowym umieszczonym w pamięci ROM. Po złożeniu komputera należało napisać dla niego oprogramowanie systemowe.

Dla tych nieustraszonych fanatyków mikrokomputerów, w 1978 roku zaczęło się ukazywać pierwsze brytyjskie czasopismo „Personal Computer World".

Dwa lata później pojawiły się pierwsze brytyjskie komputery kompletne pod względem sprzętu i oprogramowania: Atom firmy Acorn i ZX80 firmy Sinclair. Obydwa można było kupić w częściach lub zmontowane, oba też miały BASIC w pamięci ROM i były przeznaczone dla użytkowników o mniejszych kwalifikacjach.

Atom, zbudowany w oparciu o procesor 6502, z pamięcią 2 KB (z możliwością rozszerzenia do 12 KB) i z wyjściem na monochromatyczny telewizor miał funkcje graficzne (koło, prosta itd.). Wywodził się z komputerów jednoukładowych, z którymi firma Acorn pojawiła się na rynku przemysłowych urządzeń sterujących. Następnym po Atomie wyrobem tej firmy miał być BBC Micro.

Z kolei Clive Sinclair (teraz Sir Clive Sinclair) i jego firma Sinclair Research byli dobrze znani z wcześniejszych dosyć dramatycznych ingerencji na rynku tanich wyrobów elektronicznych powszechnego użytku. Ich dziełem był niesławny Black Watch — cyfrowy zegarek w częściach, oraz cała gama miniaturowego sprzętu hi-fi, produkowanego 15 lat przed Japończykami. Sinclair Research wsławił się również sprzedażą wyrobów zawierających programowane kalkulatory po 25 funtów, podczas gdy inni wyceniali je na 100 funtów.

Mikrokomputerowi ZX80 przypadł honor przełamania bariery 100 funtów. Małe pudełko z białego plastiku zawierało: procesor Z80 z pamięcią RAM o pojemności 1 KB, wbudowanym BASICIEM, sprzęgiem do telewizora i magnetofonu kasetowego. ZX80 otwierał nowy rynek mikrokomputerów powszechnego użytku. Jego następca, ZX81, oprócz BASICA ze zmiennoprzecinkową reprezentacją liczb, miał zaskakująco niską cenę 70 funtów lub 50 za model do składania. Wewnątrz ZX81 znajdowały się tylko cztery mikroukłady, natomiast ZX80 miał ich 21.

Kolorowy ZX Spectrum (w USA znany jako Timex Sinclair 2000), w 1981 roku sprzedawany za 125 funtów, w dalszym ciągu dominuje na brytyjskim rynku domowych gier komputerowych, a Clive Sinclair dawno stał się multimilionerem.

Jedynie ostatnio Commodore 64 może konkurować z ZX Spectrum pod względem liczby sprzedanych egzemplarzy i dostępnego oprogramowania.

Tymczasem firma Acorn przedstawiła BBC Micro, który powstał w związku z konkursem ogłoszonym przez sieć telewizyjną BBC na opracowanie mikrokomputera szkolnego, przeznaczonego do użytku jako pomoc w edukacyjnych programach telewizyjnych. Rywalizacja była bardzo ostra, lecz szybko okazało się, że BBC Micro ze swoją kolorową grafiką i szybkim mikroprocesorem (2 MHz) jest doskonałym komputerem na potrzeby szkół. Mimo dosyć wysokiej ceny trafiał w lukę powstałą po nieudanej próbie wprowadzenia Apple'a na rynek brytyjski i zainteresował również poważniejszych amatorów.

Niestety, firma Acorn działała zbyt wolno, podstawowe prace trwały zbyt długo, zapowiadane udoskonalenia opóźniały się i wkrótce firmę — stojącą na skraju bankructwa — przejął koncern Olivetti.

Inny czołowy producent brytyjski, ACT, zaczynał od wytwarzania oprogramowania do PET-a i importu, pracujących pod nadzorem systemu CP/M komputerów CompuThing Minimax. Wkrótce firma nabrała wiatru w żagle, sprzedając model Victora 9000 Chucka Peddle'a na rynku europejskim jako ACT Sirius, a ostatnio produkuje i sprzedaje z dużym powodzeniem kilka wersji Apricota.

Wielką Brytanię ominęła spodziewana japońska inwazja, chociaż firma Sharp próbowała wprowadzić konkurencyjny dla PET-a MZ-80K. W pewnym sensie było to duże wydarzenie. Wśród wielu dziwnych komputerów domowych, pracujących z własnymi unikalnymi systemami operacyjnymi (często różne wersje tego samego komputera były niekompatybilne między sobą), pojawił się komputer z systemem CP/M 2.2 i kompilatorami wszystkich interesujących języków (oprócz Smalltalka).

Siedem lat rozwoju mikrokomputerów w Wielkiej Brytanii złożyło się nie tylko na pasmo sukcesów. Pojawiały się na rynku i znikały z niego bardzo ciekawe, jak również i całkiem „zwariowane" komputery. Znalazł się wśród nich zasilany bateriami Newbrain z wbudowanym ekranem. Bardzo nowoczesny na owe czasy (grafika, procesor Z80, nieograniczone możliwości rozszerzenia pamięci ROM i RAM) był stawiany na równi z BBC Micro, dopóki trudności produkcyjne nie przekreśliły ambitnych planów.

Był też komputer Jupiter Ace, podobny do ZX80, ale z FORTHEM zamiast BASICA w pamięci ROM. Fakt, że FORTH nie jest językiem dla początkujących, zadecydował o niepowodzeniu całego przedsięwzięcia, chociaż do dzisiaj ten komputer ma swoich zwolenników gotowych zapłacić za niego 30 funtów w bankrutujących magazynach.

Później pojawił się Dragon 32 oparty na procesorze 6809 z myślą o systemie operacyjnym OS-9. Przez sześć miesięcy 1982 roku kupienie Dragona było dobrą lokatą kapitału. Z kolei Oric stał się najpopularniejszym mikrokomputerem domowym we Francji. Warto jeszcze wymienić Lynx — pierwszy komputer, którego BASIC miał zmiennoprzecinkowo numerowane linie.

KOMPUTERY NIEZNISZCZALNE

Prawdopodobnie nie wszyscy wiedzą, że pierwszy komputer kieszonkowy wyprodukowano i produkuje się nadal w Wielkiej Brytanii. Komputery tej klasy, np. Husky firmy Husky Computers, mają zastosowanie jako przenośne, polowe komputery naukowe i wojskowe. Mają one dość wąski krąg odbiorców, nic więc dziwnego, że Husky nie jest szeroko znany. Duże wrażenia robią pokazy odporności mechanicznej tego komputera. Pierwszy Husky wyprodukowany w 1981 roku wsławił się dzięki rozpowszechnieniu zdjęć prasowych, przedstawiających komputer rozjeżdżany przez samochód wojskowy.

Husky jest oparty na kompatybilnym z Z80 (4 MHz) mikroukładzie C800 typu CMOS, produkcji National

Semiconductor i używa systemu operacyjnego CP/M.

Projektanci Husky'ego już na początku prac podjęli bardzo ważną decyzję. Wynika ona stąd, że żadne mechaniczne urządzenie pamięci masowej nie jest niezawodne przy użytkowaniu w niesprzyjających i niebezpiecznych warunkach. W ten sposób powstał komputer przechowujący dane i programy w zachowującej zawartość po wyłączeniu zasilania pamięci RAM typu CMOS. W odpowiednim momencie zawartość pamięci wczytywana jest przez układ transmisji szeregowej.

Taka konstrukcja wymaga dużej ilości pamięci RAM, dlatego Husky ma system stronicowanej pamięci po 144 KB. Ponieważ w 1981 roku pamięci były znacznie droższe niż obecnie, powodowało to wysoką cenę jednego egzemplarza — ponad 1000 funtów. Użytkowników potrzebujących mikrokomputery tak wysokiej jakości, cena nie mogła jednak powstrzymać i firma miała dużo zamówień już od pierwszego dnia. Wśród użytkowników tych komputerów znajdują się m.in. różne towarzystwa lotnicze, wojsko, kanadyjskie towarzystwo prowadzące wyrąb lasu, lokalne spółki wodne i rada miejska Londynu (stosująca je do obliczania wydatków związanych z opieką społeczną).

HUSKY HUNTER

Najnowszy model komputera nazywa się Husky Hunter, a jego cechą znamienną są wymiary — dokładnie połowa formatu A-4, typowego dla tej klasy komputerów. Wyklucza to użycie klawiatury maszynowej. Zastosowano więc klawiaturę typu kalkulatorowego i dlatego Hunter nie ma zastosowania jako przenośny procesor tekstów.

Kolejną cechą charakterystyczną jest konstrukcja tego komputera. Jak wszystkie jego poprzedniki, Husky ma obudowę ze stopów lekkich o grubości blachy pancernej, a mimo to cały komputer waży tylko 1,2 kg. Dolna pokrywa jest mocowana za pomocą dwunastu wpuszczonych śrub z gniazdem sześciokątnym i dodatkowo zabezpieczoną uszczelką gumową przed przedostaniem się wody do wnętrza. Firma Husky Computers dopuszcza czasowe użycie komputera w warunkach deszczów tropikalnych, ale nie zaleca pracy pod wodą. W rogu wyświetlacza znajduje się wskaźnik wilgotności, sygnalizujący pojawienie się wilgoci we wnętrzu, utrzymywanym w stanie suchym przez żel krzemowy.

Cztery wymienne baterie typu AA (tzw. paluszki), wystarczające na 50 godzin pracy, znajdują się w przegródce zamykanej długą śrubą z uszczelką gumową. W układzie znajdują się dodatkowo ładowalne baterie zabezpieczające przed spadkiem napięcia, który mógłby spowodować zniszczenie przechowywanych danych i programów.

Ponieważ obudowa Huntera jest hermetycznie zamknięta, instrukcja użytkowania przewiduje pracę w warunkach nad- i podciśnieniem, np. na orbicie, w batyskafie. Mocny rzemień przymocowany obrotowo do jednej strony obudowy umożliwia wygodne przenoszenie komputera na przegubie ręki.

Do przyłączenia urządzeń we-wy jest tylko jedno złącze 25-końcówkowe typu D, wpuszczone w obudowę i dobrze uszczelnione. Ciekłokrystaliczny wyświetlacz na 40 znaków w każdej z ośmiu linii może również spełniać funkcje standardowego monitora CP/M, a w BASICU programista ma do dyspozycji matrycę 240 na 64 punktów. Wyświetlacz jest osłonięty przez plastikowe okienko grubości około 1/4 cala. Kąt pochylenia ekranu może być regulowany z klawiatury, co eliminuje zbędne otwory w obudowie.

Hunter ma 48 KB oprogramowania układowego w pamięci ROM, która zawiera m.in. program komunikacyjny, edytor tekstów, interpreter BASICA, system zarządzenia plikami zgodny z CP/M 2.2, z tą różnicą, że współpracuje on z emulatorem dysków w pamięci RAM, a nie bezpośrednio z dyskietkami.

Użytkownik może wybierać programy zawarte w pamięci ROM z menu pojawiającego się po włączeniu komputera, przyciskając jednocześnie klawisz zmiany rejestru pisma (SHIFT) i numer programu. Programy pracujące pod nadzorem CP/M uruchamia się znakiem ">". Na te programy przeznaczono 54 KB pamięci, a w pozostałych 208 KB podzielonych na 48 KB strony mieści się emulator dyskowy.

BASIC zawarty w pamięci ROM, chociaż używa tej samej pamięci co CP/M, nie pracuje pod jego nadzorem i programista ma do dyspozycji pełne całe 54 KB tej pamięci. Edytor tekstów jest pełnoekranowy, ma proste funkcje typu FIND, bardzo przydatne w redagowaniu niewielkich dokumentów. Programy operowania plikami i obsługi terminala pracują z szybkością do 4800 bitów na sekundę. Specjalny program dyskowy umożliwia przenoszenie danych z Huntera na IBM PC oraz Sirius/Victor i odwrotnie. Stale pracujący zegar (kalendarz) odmierza czas z dokładnością do dwóch sekund dziennie.

Użytkownicy twierdzą, że Hunter jest tak samo efektywny w działaniu jak większe komputery ze stacją dysków i systemem CP/M, oczywiście, pomijając cechy wynikające ze specyfiki konstrukcji, jak np. małe klawisze wymagające dokładnego przyciskania, co osobom niewprawnym sprawia pewne kłopoty. Jakość obrazu wyświetlacza nie jest gorsza od obrazu Tandy Model 100.

Przykładem zastosowania tego komputera jest program napisany w BASICU na potrzeby linii lotniczych, służący jako rejestrator kasowy napojów i przekąsek podawanych w czasie lotu, dopuszczający zapłatę w kilku walutach jednocześnie. Takie programy mogą wykonywać jeszcze inne funkcje. Pisząc programy można też używać klawiszy funkcyjnych, które mogą definiować do szesnastu rozkazów.

SPECJALNE WYKONANIE

Oprócz Huntera, firma Husky Computers produkuje kilka innych modeli komputerów o takiej samej architekturze, lecz innym specjalnym przeznaczeniu. Wszystkie mają podobną obudowę tej samej grubości, z klawiaturą membranową. Stwierdzono, że pisanie na takiej klawiaturze jest łatwiejsze po włożeniu specjalnych ochronnych rękawiczek. Oczywiście, o zastosowaniu klawiatury membranowej zadecydowało zmniejszenie liczby części mechanicznych i lepsze uszczelnienie całego komputera.

Husky SP (do celów specjalnych) ma większą pamięć (352 KB) i baterie o dwukrotnie zwiększonej żywotności. Jest przeznaczony do systemowych programów ekspertowych, wymagających dużej pamięci RAM.

Husky IS (o zwiększonym bezpieczeństwie elektrycznym) spełnia wszystkie wymagania bezpieczeństwa elektrycznego, ustanowione w Wielkiej Brytanii dla urządzeń pracujących w przemyśle petrochemicznym, naftowym, gazowniczym i wydobywczym. Zastosowany szeregowy układ sprzęgający RS-422 jest dostosowany do mniejszego napięcia niż RS-232C, co zmniejsza ryzyko wywołania pożaru.

Husky A/D (analogowo-cyfrowy) ma osiem kanałów przetwarzania analogowo-cyfrowego i jest przeznaczony do rejestracji danych z oddalonych źródeł. Dodatkową jego cechą jest modem pozwalający na obsługę komputera na odległość za pośrednictwem linii telefonicznej.

Przeznaczony na potrzeby wojska **Husky M-208** pretenduje do miana pierwszego komputera uniwersalnego, spełniającego postanowienia odpowiednich norm odporności na uderzenia, drgania i temperaturę. Twierdzi się, że tylko ten komputer przetrwałby trzecią wojnę światową. Husky M-208 wyposażono w standardowe dla wojska, 19-końcówkowe gniazdko służące komunikowaniu się z otoczeniem za pomocą transmisji szeregowej. Armia brytyjska używa tych komputerów podczas prób polowych różnych systemów broni samosterującej.

Chociaż firma Husky zaleca używanie pamięci RAM do przechowywania danych w warunkach polowych, to dostarcza również stacje dysków i magnetofony kasetowe, jako zewnętrzne pamięci masowe. System współpracuje ze stacjami 3-calowych dysków elastycznych.

Opracował MARIUSZ KUC

na podstawie BYTE, Vol. 10, No. 9, 1985

Zapożyczenia w polskim słownictwie informatycznym

Języki narodów, uczestniczących w wymianie naukowej i technicznej, wpływają na siebie. Jest faktem oczywistym, że społeczność rozwinięta pod względem naukowym lub technologicznym intensywniej oddziałuje na drugiego partnera pozostającego z nim w kontakcie. Rezultatem kontaktów między członkami różnych grup językowych są zapożyczenia obcojęzyczne.

Co do zapożyczeń, to z ich obecną koniecznością w języku polskim pogodziliśmy się od dawna. Są one skutkiem kulturalnych i cywilizacyjnych związków ze światem — pisał Z. Klemensiewicz.

Pomnażanie zasobów słownikowych dla zaspokojenia wciąż przybywających potrzeb życia polskiego w rozmaitych dziedzinach i zakresach jest nieodpartą koniecznością [5].

W tabeli 1 przedstawiono orientacyjną chronologię zapożyczeń w języku polskim (według [10]). Łacina była długo językiem nauki i międzynarodowej wymiany myśli. Wiele prac matematycznych, fizycznych, chemicznych czy medycznych publikowano jeszcze w XVII wieku po łacinie lub w przekładach łacińskich. Gdy w XVI wieku powstawała polska drukowana literatura naukowa przejęła z łaciny słowo *algoritmus* [12], które później uległo procesowi dalszej adaptacji morfologicznej, przyjmując postać *algorytm*. Dublety wyrazowe *atrybut* i *własność*, *relacja* i *wzgląd* były w częstym użyciu w słownictwie reformacji [10]. W okresie renesansu rozpowszechnił się w Polsce język włoski i przetrwał w słownictwie, w architekturze, sztuce, muzyce, wojskowości, ogrodnictwie. Wojny toczone przez Polskę w XVII wieku były przyczyną nasilenia się wpływów wschodnich. Wpływy francuskie objawiły się w szczególności w drugiej połowie XVIII wieku. Z francuskiego przejęliśmy wiele czasowników, wśród nich *adresować* i *rezydować*. Przyswoiliśmy sobie m.in. rzeczowniki *arbitraż* i *etykieta*, a także jednostki miar i wag [10]. Wiek XIX przyniósł oddziaływanie języków zaborców: niemieckiego i rosyjskiego.

Wkrótce po zakończeniu I wojny światowej zaczęły napływać do języka polskiego wyrazy angielskie. Napływ ten jeszcze się wzmocnił po II wojnie światowej, choć zmalał wpływ angielski, a zwiększył się amerykański [4]. Zapożyczenia angielskie znajdujemy w słownictwie naukowym *(holizm, laser)*, w terminologii handlu *(dumping, boom)*, polityce *(eskalacja, makkartyzm)*, turystyki *(kemping, weekend)*, sportu *(dżudo, doping)*, mody *(dżinsy, klipsy)*, muzyki *(blues, longplay)* itd. Szczególnie silne — każdy informatyk wie o tym najlepiej — jest oddziaływanie w dziedzinie słownictwa informatycznego.

Ze względu na genezę, pożyczki językowe dzieli się na wzrokowe i słuchowe oraz bezpośrednie i pośrednie [10]. Różnica pomiędzy pożyczkami wzrokowymi i słuchowymi polega na tym, że pierwsze przenikają za pośrednictwem pisma, a drugie — drogą słuchową — przez bezpośrednie kontakty z językiem — dawcą zapożyczenia. Zapożyczeniami wzrokowymi z angielskiego są m.in. wyrazy: *asembler, emulator, interpreter, sorter, terminal*. Do zapożyczeń słuchowych należy *bajt*. Wydaje się, że dzięki dużej liczbie przekazów pisanych, zapożyczenia wzrokowe są częstsze.

Jako przykład zapożyczenia pośredniego niech posłuży słowo *cyfra*, które pochodzi od arabskiego *sifr* i przez średniowieczne słowo łacińskie *cifra* zostało przejęte przez polszczyznę. Niekiedy trudno jest ustalić język, z którego bezpośrednio został zapożyczony wyraz. Często wyrazy występujące w kilku językach równocześnie mają wspólne historyczne źródło w językach europejskich najczęściej — łacińskie lub greckie. W tabeli 2 zamieszczono przykłady tego rodzaju.

Objawem wpływów języków obcych na polski są kalki, tj. takie zapożyczenia, które zostały ukształtowane na wzór wyrazów obcych, chociaż ich szata językowa wydaje się nam polska. Należą do nich kalki z angielskiego: *czas rzeczywisty* (real time), *podział czasu* (time sharing), *przerwanie wektorowe* (vectored interrupt), *punkt kontrolny* (checkpoint), *skok warunkowy* (conditional jump), *system programowania* (programming system), *wyszukiwanie informacji* (information retrieval), *złożoność obliczeniowa* (computational complexity).

Wpływ języków obcych na polski objawia się też przez nadawanie wyrazom nowych odcieni znaczeniowych pod wpływem ich obcych odpowiedników. Nowe znaczenia zyskały słowa: *adres, akumulator, etykieta, generator, monitor, pętla, prolog, rozkaz, segment, separator, skok* i inne.

Jednocześnie są w użyciu dublety. Ten sam dysygnat oznaczają *brama* i *port*, *program tłumaczący* i *translator*, *separator* i *znak rozdzielający*, *maszyna cyfrowa* i *komputer*.

Słowo komputer pochodzi od łacińskiego computare (liczyć) i w XVII wieku przez francuskie *compter* zostało przejęte przez angielski [9]. Długa była droga do zadomowienia się tego słowa w języku polskim. Najpierw używano (por. [8]) nazwy *pomoc obliczeniowa*, później *aparat matematyczny, mózg elektronowy, maszyna matematyczna, elektroniczna maszyna obliczeniowa, elektroniczna maszyna cyfrowa* i *maszyna cyfrowa*.

Natomiast bardzo łatwo przyjęło się słowo *informatyka*. Pochodzi ono od francuskiego *informatique*. Neologizm ten

Tabela 1. Chronologia zapożyczeń w języku polskim

Język \ Wiek	10	11	12	13	14	15	16	17	18	19	20
Łaciński	x	x	x	x	x	x	x	x	x		
Czeski	x	x	x	x	x	x	x				
Niemiecki				x	x	x	x			x	x
Węgierski						x	x				
Ukraiński						x	x	x			
Włoski							x	x			
Turecki							x	x			
Francuski								x	x	x	
Rosyjski										x	x
Angielski										x	x

Tabela 2. Przykłady internacjonalizmów o źródłosłowach łacińskich

Język \ Źródło	algorithmus	cōdex
Angielski	algorithm	code
Bułgarski	алгоритъм	код
Czeski	algoritmus	kód
Francuski	algorithme	code
Hiszpański	algoritmo	código
Holenderski	algoritme	code
Macedoński	алгоритам	код
Niemiecki	Algorithmus	Code
Polski	algorytm	kod
Rosyjski	алгоритм	код
Rumuński	algoritm	cod
Serbo-chorwacki	algoritam	kod
Słowacki	algoritmus	kód
Słoweński	algoritem	kod
Węgierski	algorithmus	kód
Włoski	algoritmo	codice

utworzono z dwu słów: *information* (informacja) i *automatique* (automatyka, automatyczny) [3], a pierwsze jego użycie zarejestrowano w 1962 roku [7]. Akademia Francuska w komunikacie z 6 kwietnia 1967 roku zawarła następującą definicję: *Informatyka — nauka o racjonalnym przetwarzaniu, szczególnie przez maszynę automatyczną, informacji traktowanej jako nośnik wiadomości i podstawa komunikowania się w dziedzinach technicznych, ekonomicznych i społecznych* [2]. W RFN słowo *Informatik* zostało po raz pierwszy użyte w 1968 roku w znaczeniu jak francuskie *informatique* [1]. Przedtem jednak — jak informuje H. Zemanek (por. [6]) — zastrzeżono je w nazwie firmy INFORMATIK-WERKE. Upadek firmy pozwolił mu zająć miejsce w niemieckim słownictwie naukowym.

Z całą odpowiedzialnością mogę stwierdzić, że słowo informatyka zabrzmiało w Polsce po raz pierwszy ex cathedra w październiku 1968 roku — zaświadcza W. M. Turski [13]. Użył go R. Marczyński na Ogólnokrajowym Sympozjum: *Naukowe Problemy Maszyn Matematycznych*, w Zakopanem, w dniach 20—26.10.1968 r.: *„(...) uświadomienie sobie istnienia odrębnej nauki obejmującej maszyny matematyczne, maszynową technikę obliczeniową i przetwarzanie informacji, określenie jej obszaru i powiązań z innymi naukami, a także potrzeba krótkiej i jasnej nazwy — jest dzisiaj w Polsce nakazem społecznym. Wydaje mi się, że najodpowiedniejszą nazwą dla tej dziedziny w języku polskim jest INFORMATYKA (...)"* [8].

Skutecznie wprowadzono do obiegu naukowego odpowiednik łaciński stosowany — na przykład — w tytułach takich czasopism naukowych, jak: *Acta Informatica* (1972 r.) i *Fundamenta Informaticae* (1977 r.).

Warto też zwrócić uwagę na to, że w niektórych językach zakres znaczeniowy odpowiedników graficznych słowa *informatyka* jest inny niż w języku polskim, francuskim czy niemieckim. W rosyjskim информатика oznacza informację naukowo-techniczną. Podobne znaczenie ma wyraz amerykański *informatics*, wyrażający to samo co *information science*, czyli „gromadzenie, klasyfikację, przechowywanie i rozpowszechnianie zapisanej wiedzy (...)" [14].

Były próby — nie uwieńczone powodzeniem — rozszerzenia znaczenia polskiego słowa *informatyka* przez objęcie nim także informacji naukowo-technicznej (por. Maszyny Matematyczne, nr 3. 1969, s. 26). Znalazło to wyraz w suplemencie do Wielkiej Encyklopedii Powszechnej PWN (1970), gdzie *informatyka* jest nazwą zbiorczą dwu neologizmów: *infotroniki*, dyscypliny mającej obejmować zastosowania komputerów oraz *informatoryki*, zajmującej się „teorią informacji naukowej, technicznej i ekonomicznej".

* * *

Język jest tworem społecznym i nie rozwija się w izolacji, lecz w ścisłym powiązaniu z rozwojem nauki, kultury i cywilizacji. Nawet w tak odizolowanym języku jak chiński przyjęły się m.in. słowa *algol* i *bit* [11]. Społeczność, mówiąca swym językiem narodowym stanowi rodzaj filtru, który pewne barbaryzmy przepuszcza, inne eliminuje. A że nie ma możliwości wyeliminowania nawet najbardziej dziwacznych zapożyczeń, niech zaświadczy francuskie słowo *vasistas* (lufcik, okienko), wywodzące się z niemieckiego *was ist das?*

Janusz STOKŁOSA

LITERATURA

[1] Bauer F. L., Goos G.: Informatyka. WNT, Warszawa, 1977
[2] Corge Ch.: Elementy informatyki. PWN, Warszawa, 1981
[3] Dictionnaire Encyclopedique Quillet. Librairie Quillet, Paris, 1969
[4] Grabowska I.: Nowsze zapożyczenia angielskie w języku polskim. Prace Filologiczne, Tom 23, str. 221—233, 1972
[5] Klemensiewicz Z.: Ze studiów nad językiem i stylem. PWN, Warszawa, 1969
[6] Klepacz W.: Co to jest informatyka. Informatyka, nr 6, str. 18—20, 1972
[7] Lexis. Dictionnaire de la langue française Librairie Larousse, Paris, 1975
[8] Marczyński R.: Informatyka, czyli maszyny matematyczne i przetwarzanie informacji. Maszyny Matematyczne, nr 1, str. 1—5, 1969
[9] Onions C. T. (ed.): The Oxford Dictionary of English Etymology. Clarendon Press, Oxford, 1966
[10] Rybicka H.: Losy wyrazów obcych w języku polskim. PWN, Warszawa, 1976
[11] Stoberski Z.: Międzynarodowa terminologia naukowa. PWN, Warszawa, 1982
[12] Stokłosa J.: Dixit Algorizmi, czyli o pochodzeniu i znaczeniach słowa algorytm. Informatyka, nr 11/12, 1985
[13] Turski W. M.: Nie samą informatyką. PIW, Warszawa, 1980
[14] Webster's New Collegiate Dictionary. Merriam, Springfield (MA), 1980.

Zakleszczenie

Liczba komputerów osobistych, użytkowanych obecnie w Polsce, jest dość trudna do określenia. Po pierwsze — ze względu na trudność sformułowania właściwej definicji mikrokomputera, a po drugie — ze względu na brak jakichkolwiek, nawet szacunkowych, danych cząstkowych. Przykładowo twierdzi się, że w województwie wrocławskim w prywatnych rękach jest około 60 tys. mikrokomputerów. Wiadomo także, że najbliższe plany produkcyjne firm polonijnych, spółek, przedsiębiorstw i innych zakładów zamykają się — mówiąc pesymistycznie — liczbą kilku tysięcy sztuk rocznie różnego typu komputerów osobistych w dowolnym wyborze — pod warunkiem, że będzie to wybór „koloru czarnego", czyli IBM PC. Oprócz tego istnieją i będą się rozwijać konstrukcje kopiowane lub oryginalne — oparte na sprowadzanych dowolnymi drogami mikroprocesorach.

Magia sprzętu komputerowego spędza sen z powiek tym, co próbują odpowiedzieć sobie przed zakupem na proste pytania:

- czy kupić od firmy polonijnej za miliony złotych?
- czy sprowadzić bezpośrednio z Zachodu lub z Tajwanu za dewizy?
- czy może lepiej złożyć samemu systemem gospodarczym?
- jaki typ mikroprocesora preferować (chyba najlepiej 16-bitowy)?
- ile stacji dyskietek dołączyć (oczywiście co najmniej dwie)?
- jakiej pojemności zainstalować dysk typu Winchester (oczywiście najlepiej 20 MB, a może i więcej)?
- jakiego typu drukarkę kupić?

Pytania można mnożyć, odpowiedzi są z reguły nastawione na zakup maksymalnej konfiguracji, a w końcu decyduje i tak szybkość dostawy, a przede wszystkim możliwości finansowe. Rzadko kiedy zakup jest poprzedzony rzeczową kalkulacją potrzeb i dopasowaniem konfiguracji sprzętu do rzeczywistych zastosowań.

Zanim jednak pogrążymy się w gąszczu mikrokomputerowych problemów, spróbujemy podzielić nasze zainteresowania w zależności od klasy sprzętu oraz wymagań użytkownika.

Na początku odsuniemy na bok komputerki do zabawy — typu ZX Spectrum czy nawet Spectrum Plus. Oczywiście nie neguję ich roli w edukacji i wprowadzaniu w świat informatyki, lecz z drugiej strony zadziwiają mnie niektóre ich zastosowania w pracach naukowych, badawczych lub do zakładania baz danych. Jest to całkowite nieporozumienie i tylko brak innych możliwości może usprawiedliwiać takie postępowanie. Ktoś może spytać, dlaczego tak

sądzę? Skoro komputerki te mają całkiem przyzwoity mikroprocesor oraz sporo pamięci (w porównaniu ze starymi, ale jakże wtedy mądrymi komputerami typu ZAM, UMC-1 czy GIER), to czego im brakuje? Odpowiedź zamyka się w definicji klasy sprzętu, a przede wszystkim — jego niezawodności i warunków pracy. Z nieprzekonanymi mogę zaś się zgodzić, że gokartem też można dojechać z Warszawy nad Bałtyk.

Trzeba też tutaj podkreślić, że nie pomijam przy okazji istotnych zastosowań mikroprocesorów 8-bitowych, a nawet 4-bitowych czy segmentowych, wchodzących w skład wielu bardzo pożytecznych układów do sterowania. Ta klasa zagadnień jest bardzo ważna, a często zdumienie budzi stosowanie większych konfiguracji niż to jest niezbędne.

Wróćmy jednak do zastosowań komputerów osobistych, wyposażonych co najmniej w stację dyskietek oraz ewentualnie w dysk typu Winchester. W zasadzie jest to konfiguracja wystarczająca, a czasem nawet zbyt bogata w wielu zastosowaniach. Należy sobie jednak powiedzieć, że IBM PC, Apple czy Macintosh są w swej istocie sprzętem przeznaczonym raczej do celów prywatnych lub dla bardzo małych przedsiębiorstw i instytucji. Na Zachodzie świat użytkowników sprzętu mikrokomputerowego został podzielony na część prywatną i profesjonalną. W tej drugiej części działają całkiem inne firmy i inny jest też sprzęt — o wyższym stopniu niezawodności i funkcjonalności. Dla nas ten obszar jest coraz mniej dostępny z powodów finansowych oraz embarga. Musimy sobie jednak zdawać sprawę, że dzieli nas od świata przepaść, której nawet tysiącami IBM PC nie będzie można zasypać. O komputerach średnich, dużych i superkomputerach — obiecuję już dalej nie wspominać, aby nie denerwować czytelników.

Tyle o magii sprzętu. Osobiście traktuję obecne zakupy jako typowy przykład gotowania zupy na gwoździu. Jest to bowiem jedynie maleńki — przyznaję, że bardzo kosztowny — kroczek na drodze wdrażania informatyki. O jakości i możliwościach rzeczywistego jego wykorzystania i zastosowania decyduje bowiem OPROGRAMOWANIE!

Dopiero tu rozpoczynają się kłopoty, gdyż nikt (albo bardzo niewielu — co potwierdza regułę), kto kupuje drogi sprzęt, nie zadaje sobie trudu, aby odpowiedzieć na pytanie — w jaki sposób i przy jakich pracach będzie wykorzystywał zakupiony sprzęt, gdy już nacieszy się nim dyrektor oraz hobbiści informatyki. Jeżeli zaś nawet wiadomo po co kupiono komputer, to niewiele się myśli o oprogramowaniu go do tych celów. Niestety, oprogramowanie firmowe sprzedawane z komputerem lub uzyskane drogą kolejnego kopiowania — po fascynacji posiadaniem wielu wspaniałych translatorów języków, takich jak Turbo Pascal, Aztec C, Super Basic czy nawet Prolog — nie służy niczemu konkretnemu w odniesieniu do naszych potrzeb. Te same zdania można odnieść do pozostałych programów. Programy redagujące teksty rozumieją tylko język angielski. Programy zarządzające bazami danych z trudnością dają się adaptować, a pakiety programów graficznych są fascynujące — tylko co by tu narysować? A tak naprawdę, to brakuje porządnego polskiego edytora, programu fakturowania czy obsługi księgowości oraz magazynu.

W rezultacie, w setkach instytucji stoi sprzęt o niebagatelnej już wartości — w praktyce bezużyteczny. Sprzęt ten nie zarabia na swoją amortyzację i coraz bardziej kłuje w oczy księgowych. W Polsce nie istnieje bowiem rynek oprogramowania, które spełnia następujące warunki:

- konwersacja programu z użytkownikiem odbywa się w języku polskim, w słownictwie czytelnym dla sekretarki, magazyniera i księgowego
- wprowadzane przez program dokumenty mają postać dobrze znanych formularzy i druków
- przechowywanie i obieg dokumentów w systemie programów jest zgodny z istniejącymi w Polsce przepisami i pragmatyką urzędniczą itd.

Tego wszystkiego jak na razie nie ma i poza incydentalnymi przypadkami długo nie będzie. Co więcej, różne instytucje zmuszone jakoś — to odpowiednie w tym wypadku słowo — zagospodarować swój sprzęt, decydują się na samodzielne tworzenie takiego oprogramowania. Rozpoczyna się kosztowna zabawa w powstawanie różnego rodzaju programów użytkowych, projektowanych ad hoc, bez głębszego przemyślenia i koniecznego doświadczenia. Typowe jest marnowanie energii na równolegle prowadzone prace nad tymi samymi tematami przez różne instytucje. W najbliższym czasie oczekuję więc powstania kilkunastu programów redagujących teksty (pomijając przekład Wordstara na język polski), programów fakturujących oraz księgujących. I tylko na początku wydaje się, że to nic trudnego napisać takie programy — nie potrzeba już przecież kupować sprzętu, a wystarczy tylko mieć pomysły, czas i trochę drobnych pieniędzy. Ba! Wkrótce okazuje się, że tego czasu i nakładów finansowych potrzeba coraz więcej. Nie u nas odkryto, że wartość oprogramowania ponad trzykrotnie, a niekiedy nawet dziesięciokrotnie przekracza wartość sprzętu. Być może, u nas te relacje wynagrodzeń od kosztów sprzętu są nieco zakłócone, ale o tych „parę milionów" nie będziemy się kłócić.

Obecnie mamy sytuację typową dla dziedziny programowania współbieżnego — zwaną zakleszczeniem — powodującą wstrzymanie działania programu wskutek wzajemnego oczekiwania procesów współbieżnych na zdarzenia, których spełnienie jest wzajemnie uwarunkowane. Dalsze sprowadzanie sprzętu, chociaż możliwe, będzie ograniczane brakiem motywacji do jego zakupu, ze względu na niemożliwość zastosowania z powodu braku oprogramowania. Z kolei, wytworzenie użytecznego oprogramowania wymaga powołania silnych zespołów wyposażonych w dobry sprzęt oraz spore możliwości finansowe. Problem zaś leży w tym, kto ma dokonać takiej inwestycji — czy aktualni posiadacze sprzętu, którzy wydali już pieniądze, czy przyszli posiadacze, którym wtedy zabraknie na sprzęt. Rozwiązanie jak zawsze leży pośrodku, ale konieczne jest sprzężenie wysiłków, gdyż pojedynczy właściciele komputerów nie będą w stanie podołać tym wymaganiom finansowym. Z drugiej strony wiemy, jak trudno jest handlować oprogramowaniem, gdyż koszty jego opracowania, z powodu braku realnej ochrony przed kopiowaniem, trudno jest rozłożyć na kolejnych klientów.

Cóż więc pozostaje? Moim zdaniem należy propagować rzetelne, dobre i użyteczne oraz sprawdzone rozwiązania krajowe, zmierzające do praktycznego zastosowania posiadanego sprzętu komputerowego. Publikowanie opisu takich rozwiązań z prezentacją podstawowych idei struktury algorytmów umożliwi ochronę wynikającą z prawa autorskiego i będzie stanowić podstawę do roszczeń w stosunku do nieuczciwych. Jednocześnie publikacje te mogą ułatwić wymianę doświadczeń i ograniczenie dublowania prac i wysiłków, a w rezultacie — doprowadzić do szybkiego nasycenia rynku podstawowymi własnymi programami. Liczba takich programów jest przecież w końcu ograniczona.

Oczekując od Czytelników opinii w tej sprawie — Redakcja Informatyki udostępnia swoje łamy dla prezentacji praktycznego wykorzystania sprzętu mikrokomputerowego w oparciu o własne opracowania.

WACŁAW ISZKOWSKI

Redakcja INFORMATYKI udostępnia łamy czasopisma na prezentację praktycznych i sprawdzonych zastosowań sprzętu mikrokomputerowego oraz krajowego oprogramowania użytkowego.

W naszym rozumieniu sprzętem mikrokomputerowym jest komputer osobisty, wyposażony co najmniej w stację dyskietek.

Preferujemy opracowania przedstawiające oryginalne, sprawdzone rozwiązania — możliwe do szerszego zastosowania.

Forma tych prac może być dowolna:

- **artykuły** — opisujące ideę oraz metody rozwiązania problemu z prezentacją istotnych elementów algorytmów
- **notatki** — opisujące zalety funkcjonalne produktu programowego w określonych zastosowaniach
- **ogłoszenia** (płatne przez producenta) — reklamujące programy i wdrożenia sprzętu mikrokomputerowego wraz z warunkami ich dystrybucji.

amepol
PRZEDSIĘBIORSTWO ZAGRANICZNE

Przedsiębiorstwo Zagraniczne AMEPOL

oferuje

PÓŁPRZEWODNIKOWE PAMIĘCI OPERACYJNE

do komputerów:

- ODRA 1305

pojemność od 64 k do 2 M

- R – 32

pojemność od 256 k do 16 MB

do komputerów:

- MERA 400

pojemność od 64 k do 1 M

- MERA 9150 (SEECHECK)
- PDP 11 – SM 4

Pamięci wykonane są na elementach VLSI produkcji zachodniej.
Na powyższe pamięci udzielamy dwuletniej gwarancji.
Terminy dostawy według życzeń klienta.

PZ AMEPOL oferuje ponadto:

- do komputera ODRA 1305 skaner oraz multiplekser
- do minikomputera MERA 400 procesory peryferyjne PLIX i MULTIX oraz moduł inicjatora

Szczegółowych informacji udziela:

Przedsiębiorstwo Zagraniczne
AMEPOL

Zakład Elektroniki i Aparatury Medycznej
Plac Żelaznej Bramy 1
00-136 Warszawa
tel.: 20-34-75, 20-45-05
teleks: 812539 apol pl

amepol

EO/461/K/86

Bjorner D.: Rola i zakres formalnej definicji Ady (1)

INFORMATYKA 1986, nr 5, s. 1

Pierwsza część charakterystyki problemu formalnej definicji języka Ada. Omówiono kwestie językowe, grupy użytkowników specyfikacji języka oraz techniki specyfikacji.

Perycz K.: Mikrokomputerowy system operacyjny S1 (1)

INFORMATYKA 1986, nr 5, s. 5

Pierwsza część charakterystyki nowego uniwersalnego systemu operacyjnego S1 dla mikrokomputerów. Omówiono procesory poleceń, najważniejsze polecenia oraz listy poleceń.

Michalik K.: Programowanie logiczne

INFORMATYKA 1986, nr 5, s. 7

Charakterystyka teoretycznych podstaw programowania logicznego. Omówiono na przykładach klauzule Horna, sposoby wnioskowania i rozwiązywania problemów oraz konstrukcję programów logicznych.

Cofta P.: Norma IEEE — języki assemblerowe dla mikroprocesorów

INFORMATYKA 1986, nr 5, s. 11

Szczegółowe omówienie treści nowej normy IEEE 694-1985 przeznaczonej do wspomagania projektowania języków asemblerowych dla mikroprocesorów.

Karczmarczuk J.: Język programowania Icon (3)

INFORMATYKA 1986, nr 5, s. 14

Dokończenie charakterystyki języka Icon, obejmujące dalsze omówienie generatorów oraz sposób realizacji operacji tekstowych.

Grzegorzewicz P.: Mechanizm półautomatycznego generowania poleceń dla systemu operacyjnego RSX-11M

INFORMATYKA 1986, nr 5, s. 17

Charakterystyka półautomatycznego generowania poleceń w systemie operacyjnym RSX-11M. Omówiono sposób generowania poleceń oraz zalety i ograniczenia metody.

Macioł A., Stawowy A.: Środki do przetwarzania tekstów — historia i dzień dzisiejszy

INFORMATYKA 1986, nr 5, s. 20

Definicja pojęcia, klasyfikacja i charakterystyka oraz ocena różnych urządzeń do przetwarzania tekstów. W zakończeniu omówiono problem przetwarzania tekstów w Polsce.

Бьернер Д.: Роль и и объем формальной дефиниции языка Ада (I)

INFORMATYKA 1986, № 5, с. 1

Первая часть характеристики проблемы формальной дефиниции языка Ада. Вопросы, касающиеся языка, группы пользователей спецификации языка и техник спецификации.

Перыч К.: Операционная система 91 для микро-ЭВМ (1)

I
NFORMATYKA-1986, № 5, с. 5

Первая часть характеристики новой универсальной операционной системы 91 для микро-ЭВМ. Описание процессоров команд, важнейших команд и списка команд.

Михалик К.: Логическое программирование

INFORMATYKA 1986, № 5, с. 7

Характеристика теоретических основ логического программирования. Описание на пример2х оговорки Хорна, способы внесения и решения проблем, а также конструкции логических программ.

Кофта П.: Стандарт IEEE — языки ассемблера для микропроцессоров

INFORMATYKA 1986, № 5, с. 11

Подробное описание содержания нового стандарта IEEE 694—1985, предназначенного для облегчения проектирования языков ассемблера для микропроцессоров.

Карчмарчук Е.: Язык программирования Икон (3)

INFORMATYKA 1986, № 5 с. 14

Завершение характеристики языка Икон, включающее очередное описание генераторов и способ реализации операций на текстах.

Гжегожевич Р.: Механизм полуавтоматического генерирования команд для опредационной системы РСХ-11М.

INFORMATYKA 1986, № 5, с. 17

Характеристика полуавтоматического генерирования команд в операционной системе РСХ-11М. Описание способа генерирования команд, достоинства и ограничения этого метода.

Мациол А., Ставовы А.: Средства для обработки ткестов — история и сегодняшний день

INFORMATYKA 1985, № 5, с. 20

Определение понятия, классификаци0 и характеристика, а также оценка разичных устройств для обработки текстов. В заключение обсуждается вопрос обработки текстов в Польше.

Bjørner D.: Role and scope of Ada formal definition (1)

INFORMATYKA 1986, No. 5, p. 1

First part of the characteristices of the Ada formal definition. Language issues, users of a language specification and specification techniques are discussed.

Perycz K.: S1 - a microcomputer operating system (1)

INFORMATYKA 1986, No. 5, p. 5

First part of the characteristices of the S1 - a new general-purpose operating system for microcomputers. Command processors, main commands and command's lists are discussed.

Michalik K.: Logic programming

INFORMATYKA 1986, No. 5, p. 7

Characteristics of the theoretic foundations of logic programming. Horn clauses, problem aolutions, as well as structure of logical programs, are discussed.

Cofta P.: IEEE Standard — assembly languages for microcomputers

INFORMATYKA 1986, No. 5, p. 11

Detailed discussion of the new IEEE Standard 694-1985 which is devoted to microprocessor assembly languages.

Karczmarczuk J.: Icon programming language (3)

INFORMATYKA 1986, No. 5, p. 14

Final part of Icon characteristics which includes a continuation of generators presentation, as well as a description of text operations.

Grzegorzewicz P.: Mechanism of semiautomatic command generation for the RSX-11M operating system

INFORMATYKA 1986, No. 5, p. 17

Characteristics of semiautomatic command generation for the RSX-11M operating system. The way, of command generation as, well as advantages and limitations of the method, are discussed.

Macioł A., Stawowy A.: Text processing — history and state of the art.

INFORMATYKA 1986, No. 5, p. 20

Definition, classification and characteristics, as well as evaluation of different devices for text processing. Problems of text processing in Poland are discussed.

Bjørner D.: Rolle und Bereich der formellen Definition von Ada (1)

INFORMATYKA 1986, Nr. 5, S. 1

Erster Teil einer Charakteristik des Problems von formellen Definition der Ada-Sprache. Es wurden Sprachefragen, Benutzergruppen von der Sprachedefinition, sowie Spezifikationsmethoden besprochen.

Perycz K.: S 1-Betriebssystem (1)

INFORMATYKA 1986, Nr. 5, S. 5

Erster Teil einer Charakteristik von neuen universellen Betriebssystem für Mikrocomputer. Es wurden Befehlsprozessoren, wichtigste Befehle, sowie Befehlslisten, besprochen.

Michalik K.: Logische Programmierung

INFORMATYKA 1986, Nr. 5, S. 7

Eine Charakteristik der theoretischen Grundlagen von der logischen Programmierung. Es wurden die Horn-Klausel mit Anwendungsbeispielen, Problemfolgerung und -Lösung, sowie Struktur der logischen Programme, besprochen.

Cofta P.: IEEE-Norm — Assemblersprachen für Mikroprozessoren

INFORMATYKA 1986, Nr. 5, S. 11

Eine ausführliche Besprechung des Inhaltes von neuen 694-1985 IEEE-Norm, die für Unterstützung der Assemblersprachenprojektierung von Mikroprozessoren bestimmt ist.

Karczmarczuk J.: Icon-Programmiersprache (3)

INFORMATYKA 1986, Nr. 5, S. 14

Beendigung der Icon-Sprachecharakteristik, die weitere Besprechung von Generatoren, sowie Realisierungsweise der Textoperationen, umfasst.

Grzegorzewicz P.: Ein Mechanismus der halbautomatischen Generierung von Befehlen für das RSX-11M-Betriebssystem

INFORMATYKA 1986, Nr. 5, S. 17

Eine Charakteristik der halbautomatischen Generierung von Befehlen im RSX-11M-Betriebssystem. Es wurden Arbeitsweise der Befehlsgenerierung, sowie Vorteile und Begrenzungen der Methode, besprochen.

Macioł A., Stawowy A.: Textverarbeitungsmittel — Geschichte und Gegenwart

INFORMATYKA 1986, Nr. 5, S. 20

Definition des Begriffes, Klassifikation und Charakteristik, sowie Beurteilung von verschiedenen Einrichtungen für Textverarbeitung. In der Schlussteil wurde das Problem der Textverarbeitung in Polen besprochen.

COMPUTER STUDIO KAJKOWSCY
ul. Balladyny 3B, 81-524 Gdynia
Tel.: 29-00-18, 24-01-50

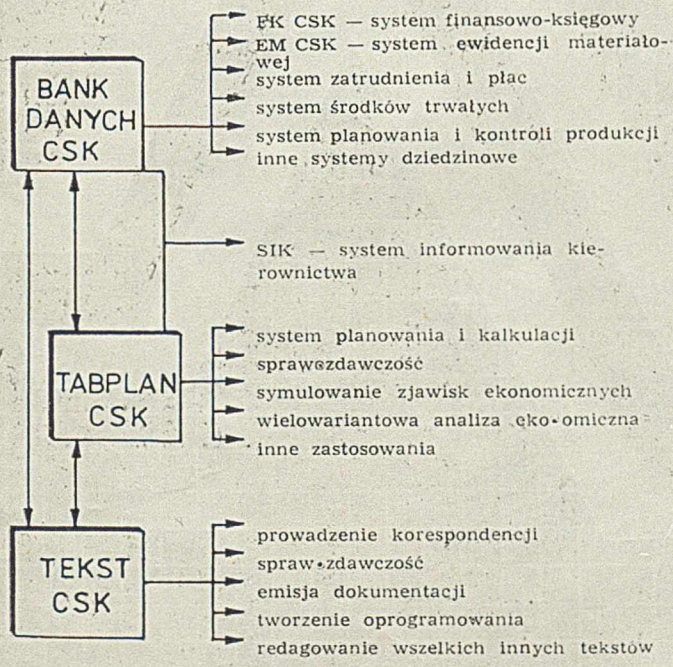
Oprogramowanie z CSK dla potrzeb zarządzania

(dla mikrokomputerów 8- i 16-bitowych)

Oferta programowa CSK obejmuje programy, które służą do automatyzacji przetwarzania danych we wszystkich dziedzinach funkcjonowania przedsiębiorstw:

- do prowadzenia korespondencji, ewidencji i rozliczeń finansowych,
- do wspomagania decyzji na szczeblu dyrektora.

Miejsce oprogramowania CSK w zautomatyzowanym systemie przetwarzania danych w przedsiębiorstwie przedstawia rysunek.



PROFESJONALNE OPROGRAMOWANIE MIKROKOMPUTERÓW

CSK oferuje następujące oprogramowanie użytkowe i systemowe dla mikrokomputerów 8- i 16-bitowych

Nazwa programu	8-bitowe	16-bitowe
Oprogramowanie użytkowe		
BANK-DANYCH CSK — system zarządzania bazą danych	+	+
TABPLAN CSK — komputerowy arkusz kalkulacyjny	+	+
TEKST CSK — pakiet redagowania tekstów	+	+
TRANSCOM CSK — program komunikacji z ODRĄ	+	+
TRANSCOM/M CSK — program komunikacji między mikrokomputerami	+	+
BANK-CSK — graficzny system komunikacji z bazą danych	—	+
BGRAF CSK — pakiet grafiki prezentacyjnej	—	+
FK CSK — system finansowo-księgowy	+	+
EM CSK — system ewidencji materiałowej	+	+
PL TEKST CSK — pakiet redagowania tekstów (polskie znaki)	—	+
Oprogramowanie systemowe		
SOMIK — rozbudowa systemu operacyjnego CP/M 2.0	+	—
W/SYS CSK — system operacyjny wielozadaniowy/wielostanowiskowy	+	+
GSK CSK — pakiet procedur graficznych wg normy GKS	—	+

+ tak; — nie

Oprogramowanie CSK może być eksploatowane **na mikrokomputerach 8-bitowych:**

- ELWRO seria 500 i 600
- ROBOTRON 5110/20/30, 1715
- MK 4101/02
- ComPAN
- IMP-85
- innych (z systemem CP/M)

mikrokomputerach 16-bitowych:

— LIDIA II/XT
— MAZOWIA
— ELWRO 800
— M24 (Olivetti)
— innych (zgodnych z IBM PC XT i AT).

Cena obejmuje: dyskietki z programem (wraz z kopią), jeden egzemplarz dokumentacji użytkowej. Termin realizacji — 7 dni od daty otrzymania zamówienia.
CSK organizuje kursy użytkowania oprogramowania.

EO/81/K/86

KOVO

Przedsiębiorstwo Handlu Zagranicznego
PZO KOVO PRAGA CZECHOSŁOWACJA
ZAPRASZA
NA POKAZ „KOVO – WARSZAWA 1986"
KTÓRY ODBĘDZIE SIĘ W DNIACH
OD 2. DO 6. 9. 1986
W POLITECHNICE WARSZAWSKIEJ
PLAC JEDNOŚCI ROBOTNICZEJ 1
AULA GŁÓWNA WARSZAWA

■ PRZYRZĄDY LABORATORYJNE ORAZ POMIAROWE ■ TECHNIKA BIUROWA ORAZ OBLICZENIOWA ■ MASZYNY POLIGRAFICZNE, KOPIARKI ORAZ POWIELACZE ■ LABORATORIA, TECHNIKA LABORATORYJNA ORAZ JĄDROWA ■ PRZYRZĄDY AUTOMATYZACJI PRZEMYSŁOWEJ, REGULACJI ORAZ POMIARÓW ■ ELEMENTY ELEKTRONICZNE BIERNE ORAZ CZYNNE, URZĄDZENIA TECHNOLOGICZNE DLA ELEKTRONIKI, ŹRÓDŁA ŚWIATŁA ■ PRZYRZĄDY I URZĄDZENIA TELEKOMUNIKACYJNE

EXPORT
IMPORT
KOVO
PRAHA
CZECHOSLOVAKIA